Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 61 (2016)

Constant-Round Interactive Proofs for Delegating Computation*

Omer Reingold† Stanford University
Guy N. Rothblum‡ Weizmann Institute
Ron D. Rothblum§ Technion

October 14, 2020

Abstract

The celebrated IP = PSPACE Theorem [LFKN92, Sha92] allows an all-powerful but untrusted prover to convince a polynomial-time verifier of the validity of extremely complicated statements (as long as they can be evaluated using polynomial space). The interactive proof system designed for this purpose requires a polynomial number of communication rounds and an exponential-time (polynomial-space complete) prover. In this paper, we study the power of more efficient interactive proof systems.

Our main result is that for every statement that can be evaluated in polynomial time and bounded-polynomial space there exists an interactive proof that satisfies the following strict efficiency requirements: (1) the honest prover runs in polynomial time, (2) the verifier is almost linear time (and under some conditions even sub linear), and (3) the interaction consists of only a *constant number of communication rounds*. Prior to this work, very little was known about the power of efficient, constant-round interactive proofs (rather than arguments). This result represents significant progress on the round complexity of interactive proofs (even if we ignore the running time of the honest prover), and on the expressive power of interactive proofs with polynomial-time honest prover (even if we ignore the round complexity). This result has several applications, and in particular it can be used for verifiable delegation of computation.

Our construction leverages several new notions of interactive proofs, which may be of independent interest. One of these notions is that of *unambiguous interactive proofs* where the prover has a unique successful strategy. Another notion is that of *probabilistically checkable interactive proofs*[1] (PCIPs) where the verifier only reads a few bits of the transcript in checking the proof (this could be viewed as an interactive extension of PCPs).

*A preliminary version appeared as [RRR16]. This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

†Email: omer.reingold@gmail.com. Parts of this work were done while at Samsung Research America.

‡Email: rothblum@alum.mit.edu. Parts of this work were done while at Samsung Research America.

§Email: rothblum@cs.technion.ac.il. This work was done in part while the author was at MIT. Supported by grants NSF MACS - CNS-1413920, DARPA IBM - W911NF-15-C-0236, and SIMONS Investigator award Agreement Dated 6-5-12.

[1]An equivalent notion to PCIPs, called *interactive oracle proofs*, was recently introduced in an independent work of Ben-Sasson *et al.* [BCS16].

ISSN 1433-8092

Contents

1 Introduction

The power of efficiently verifiable proof systems is a central question in the study of computation. In this work, we study the power of *Interactive Proof systems* [GMR89, BM88]. An Interactive Proof system is an interactive protocol between a randomized verifier and an untrusted prover. The prover convinces the verifier of the validity of a computational statement, usually framed as the membership of an input x in a language $\mathcal{L}$. Soundness is unconditional. Namely, if the input is not in the language, then no matter what (unbounded and adaptive) strategy a cheating prover might employ, the verifier should reject with high probability over its own coin tosses. Interactive proofs have had a dramatic impact on Complexity Theory and Cryptography. The celebrated $\mathsf{IP} = \mathsf{PSPACE}$ Theorem [LFKN92, Sha92] showed that interactive proofs are remarkably powerful: a polynomial-time verifier can use them to verify any statement/language that is computable in polynomial space. This remarkable characterization opened the door to foundational questions about the power and the complexity of interactive proofs.

The Complexity of Verifying and Proving. The [LFKN92, Sha92] protocol places a heavy burden on the honest prover, who needs to perform intractable computations. Indeed, this was unavoidable as they focused on interactive proofs for statements (in PSPACE) that are themselves intractable. A study of interactive proofs for *tractable* statements, where both the honest prover and the verifier run in polynomial time, was initiated in [GKR08, GKR15]. The verifier should be *super-efficient*, e.g. it should run in linear or near-linear time. In particular, verification requires significantly less resources than it would take to decide the (tractable) language. We refer to interactive proofs with an efficient (polynomial-time) honest prover, and a super-efficient verifier as *doubly-efficient* interactive proof systems. We emphasize that we still require unconditional soundness against an unbounded cheating prover. The first question that we study in this work is:

Question 1.1. *Which languages have* doubly-efficient *interactive proofs?*

The *Round Complexity* of Interactive Proofs. The [LFKN92, Sha92] protocol can be used to prove membership in any language $\mathcal{L}$ that can be decided in space $S = S(n)$, with communication complexity that is polynomial in S, and verification time that is roughly linear in n and polynomial in S. To achieve the small communication and verification time, the prover and the verifier need to engage in $\mathsf{poly}(S)$ rounds of communication. It is natural to ask whether the number of rounds can be reduced. The second question we study is:

Question 1.2. *Which languages have* constant-round *interactive proofs with small communication and verification time?*

Even for protocols with exponential-time honest provers, this question is not well understood.

1.1 Our Main Result

We make simultaneous progress on both of these foundational questions. Our main result is a new construction of *doubly-efficient* and *constant-round* interactive proofs for languages that are computable in polynomial time and bounded space.

Theorem 1 (Doubly-Efficient Interactive Proofs for Bounded Space)**.** *Let $\mathcal{L}$ be a language that can be decided in time $\mathsf{poly}(n)$ and space $S = S(n)$, and let $\delta \in (0, 1)$ be an arbitrary (fixed) constant.*

There is a public-coin interactive proof for $\mathcal{L}$ with perfect completeness and soundness error $1/2$. The number of rounds is $O(1)$. The communication complexity is $(\mathsf{poly}(S) \cdot n^{\delta})$. The (honest) prover runs in time $\mathsf{poly}(n)$, and the verifier runs in time $(\tilde{O}(n) + \mathsf{poly}(S) \cdot n^{\delta})$.

Furthermore, if the verifier is given query access to a low-degree extension of the input, then its running time is reduced to $O(\mathsf{poly}(S) \cdot n^{\delta})$.

In particular, we obtain an (almost) linear-time verifier and a polynomial-time honest prover for languages computable in polynomial time and bounded-polynomial space. Here and below, by *bounded-polynomial space* we mean space n^{σ} for some sufficiently small universal constant $\sigma > 0$. See Corollary 8 in Section 5 for a more formal and general statement.

In several ways, the result of Theorem 1 seems tight. The dependence on the space S in the communication or the verification time is tight (up-to polynomial factors), as any language that has an interactive proof, where the communication and verification time (or rather verification space) are bounded by $B \geq \log n$, can be decided in space $\mathsf{poly}(B)$ (this can be seen from the game tree construction of Goldreich and Håstad [GH98]). Also, under reasonable complexity conjectures, no constant-round interactive proof for bounded space computations can have sub-polynomial communication complexity, as this would lead to a super-polynomial AM-speedup for that class of computations (see Remark 5.1 for further details). We also emphasize that the result of Theorem 1 is unconditional, we make no cryptographic assumptions, and the interactive proof is sound even against unbounded cheating provers.

Applications: Delegation, Succinct Zero-Knowledge, IPPs, and more. Beyond their importance in the theoretical study of computation, interactive proofs are also motivated by real-world applications, such as delegating computation. Here, a powerful server can run a computation for a weak client, and provide an interactive proof of the output's correctness, see [GKR15]. The interactive proof should be doubly-efficient, so that generating the proof is tractable for the server, and verification is super-efficient for the client. Naturally, this scenario focuses on tractable computations that can actually be performed by the server. The interactive proofs of Theorem 1 can be used to delegate bounded-space polynomial-time computations (without making computational assumptions or using cryptographic machinery). The constant round complexity is an important bonus in the delegation scenario, where network latency can make communication rounds expensive.

Given the fundamental importance of interactive proofs in cryptography and complexity, it should come as no surprise that Theorem 1 has implications to a variety of foundational questions. It implies constant-round *succinct* zero-knowledge proofs from one-way functions for any NP-language whose witnesses can be verified in bounded space. It also gives constant-round sublinear-time Interactive Proofs *of Proximity* for polynomial-time and bounded-space computations. Finally, Theorem 1 can be used to extend the class of circuits to which the [GKR15] protocol applies, from $\mathsf{Logspace}$-uniformity to general bounded-space uniformity. We elaborate on these implications in Section 1.2.

A Taste of the Techniques. The overall structure of our proof of Theorem 1 is to iteratively construct a proof system for longer and longer computations. Assume that we already have an interactive proof for verifying Turing machine computations that run in time T and space S, we now want to extend the proof system to verifying computations that run in time $(k \cdot T)$ and space S, for some super-constant integer k. This could easily be reduced to verifying k computations that run in time T (all the prover needs to do is to send is $k-1$ intermediate states of the machine).

Two simple but ineffective approaches are to either run k instances of the "base" proof system to verify the k computations (which is inefficient) or to spot-check a few of the computations (which drastically increases the success probability of a cheating prover, known as the soundness error). The main ingredient of our proof is therefore what we call a *batch verification theorem* for interactive proofs. Such a theorem allows the verification of k computations in a much more efficient way than k independent executions (and while maintaining the soundness error).

To obtain the batch verification theorem, we introduce several new notions for interactive proofs that may be of independent interest. First we suggest a notion where the prover has a unique strategy to convince a verifier (similarly to the unique satisfying assignment of a unique-SAT formula). The moment the prover deviates from the prescribed strategy it will likely fail in convincing the verifier *even when the statement in question is true*. We call this notion an *unambiguous interactive proof*. We also introduce a notion that can be thought of as an interactive analogue of PCPs. These are interactive proof systems where the verifier only reads a few bits of the input and transcript in checking the proof. We call these proofs *probabilistically checkable interactive proofs* (PCIPs).[2] We consider various ways to combine these notions into unambiguous PCIPs (which play an important role in this work).

We are able to provide a batch verification theorem for PCIPs (that are somewhat unambiguous). Coming to apply this theorem repeatedly, we encounter the following problem: the batch verification degrades the query complexity and verification time by a multiplicative factor of k. After repeated batching, the query complexity will no longer be sublinear in the transcript length, which is problematic. To resolve this problem, the iterative construction of our interactive proofs repeatedly uses the PCIP-batch verification step, followed by a PCIP-"query-reduction" step (which also reduces the verifier's runtime). Note that the "query-reduction" does increase the communication and so this iterative construction constantly balances verifier's query and runtime against communication complexity, while gradually obtaining powerful PCIPs (and interactive proofs) for longer and longer computations. This is similar in spirit to the delicate balancing of parameters in the iterative constructions of [RVW00, Rei08, Din07] (also see [Gol11]).

Comparison with Prior Work on Interactive Proofs. The results that are perhaps most directly related to ours are the works of [LFKN92, Sha92, GKR15], which also construct interactive proofs for rich classes of computations. For $\mathsf{poly}(n)$-time space-S languages, the protocol of [LFKN92, Sha92] has honest prover runtime $n^{\mathsf{poly}(S)}$ and $\mathsf{poly}(S, \log n)$ rounds. The communication complexity is $\mathsf{poly}(S, \log n)$ and the verifier runs in time $\tilde{O}(n) + \mathsf{poly}(S, \log n)$. The communication and verification time are tight (up to polynomial factors, see Remark 5.1). The protocol of [GKR15] gives an improved interactive proof, where the prover runs in time $\mathsf{poly}(n) \cdot 2^S$, but still has round complexity $\mathsf{poly}(S, \log n)$ (see [Rot09]).

We compare these results with the statement of Theorem 1. For languages computable in time $\mathsf{poly}(n)$ and space n^σ, Theorem 1 improves the prover running time exponentially, from $2^{O(n^\sigma)}$ to $\mathsf{poly}(n)$. The round complexity is improved from $n^{O(\sigma)}$ to $O(1)$. Even for smaller space, where past works achieve tight communication and verification times, Theorem 1 gives the first interactive proofs with a constant number of rounds (and is tight under reasonable conjectures, see Remark 5.1). For languages that are computable in $\omega(\log(n))$-space and polynomial time, Theorem 1 gives the first interactive proofs with a polynomial-time honest prover.

[2] A notion which is equivalent to PCIPs, called "Interactive Oracle Proofs", was introduced in a recent independent work of Ben-Sasson *et al.* [BCS16] (see also [BCGV16, BCG+16]). See Section 1.3 for further details on their work.

The work of [GKR15] gives doubly-efficient interactive proofs for languages computable by (Logspace-uniform) polynomial-size circuits of bounded depth $D = D(n)$. The verifier's work is quasi-linear in its input length and polynomial in the circuit depth. The communication is polynomial in the circuit depth. The number of rounds in their protocol is $O(D \cdot \log n)$. A modification to the [GKR15] protocol gives an $O(1)$-round interactive proof with n^{δ} communication for languages computable by NC^1 circuits satisfying a (very) strong notion of uniformity [KR09]. Theorem 1 can be used to extend the [GKR15] protocol to n^{σ}-space uniform polynomial-size circuit families of bounded depth. See Section 1.2.

Comparison with Prior Work in Related Models. In Section 1.3 we compare our results with what is known in models that are closely related to interactive proofs, such as PCPs, computationally sound interactive proofs and interactive proofs of proximity. Regarding PCPs, we emphasize that they provide a weaker notion of soundness than interactive proofs. Recall that a PCP is a proof system in which the verifier reads only a few bits from a proof that is fixed a priori. Soundness means that the verifier rejects alleged proofs for false statements, but the proof has to be *written down ahead of time* before the verifier's queries are specified. In particular, this static notion of soundness is not directly suitable for delegating computations over a network. For further details see Section 1.3.

Organization. In **Section 1.2** we discuss applications of Theorem 1 to the construction of succinct zero-knowledge proofs, interactive proofs of proximity, interactive proofs for languages computed by a richer class of uniform bounded-depth circuits and interactive proofs for randomized computations. We give an overview of our approach, techniques and technical contributions in **Section 2**. Preliminaries and technical definitions are in **Section 3**. New notions of probabilistically checkable and unambiguous interactive proof systems play a central role in our construction, and these are discussed and defined in **Section 4**. A general and formal statement of our main result is in **Section 5**. The subsequent sections contain the technical details of our main construction (see the roadmap in **Section 5.1**).

1.2 Further Applications

Succinct Zero-Knowledge Proofs. We use Theorem 1 to construct succinct *constant round* zero-knowledge proofs for NP statements from one-way functions. These are zero-knowledge proofs where the communication is nearly-linear in the length of the statement's NP witness. Several works constructed succinct zero-knowledge proofs for various restrictions of NP. The restrictions are obtained by examining, for an NP language $\mathcal{L}$, the polynomial-time relation $\mathcal{R}$ that checks the validity of witnesses. Succinct zero-knowledge proofs for NP-languages where $\mathcal{R}$ is computable by AC^0 circuits were shown by Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS07] and by Kalai and Raz [KR08]. This was extended in [GKR15] to NP-languages where $\mathcal{R}$ is computable by a uniform bounded-depth circuit (say of small fixed polynomial depth). This improvement was significant, especially because many statements regarding cryptographic operations cannot be directly computed or verified in AC^0 (e.g. pseudo-random functions). Their zero-knowledge proofs, however, require a number of rounds that grows with the circuit-depth of $\mathcal{R}$. Theorem 1 directly implies succinct *and constant-round* zero-knowledge proofs when $\mathcal{R}$ is computable in bounded space.

Theorem 2. *Assume one-way functions exist, let $\delta \in (0,1)$ be a constant, $\kappa = n^{\delta}$ be a cryptographic security parameter. Let $\mathcal{L}$ be an* NP *language, whose relation $\mathcal{R}$ can be computed by a* $\mathsf{poly}(n)$*-time and $O(n^{\delta})$-space Turing Machine that operates on inputs of length n and witnesses of length $m = \mathsf{poly}(n)$. The language $\mathcal{L}$ has a public-coin zero-knowledge interactive proof with perfect completeness, constant soundness, and communication $m \cdot \mathsf{poly}(\kappa)$. The (honest) prover, given a valid witness, runs in time* $\mathsf{poly}(n)$*. The verifier runs in time $(m+n) \cdot \mathsf{poly}(\kappa)$.*

Proof Sketch. The proof uses the standard transformation from public-coin interactive proofs to zero-knowledge proofs [BGG+88], and follows along the same lines as the succinct zero-knowledge construction of [GKR15]. The prover commits to the witness w using a statistically binding commitment [Nao91, HILL99], and sends this commitment to the verifier. The prover and verifier then run the *public coin* interactive proof of Theorem 1 to verify that $\mathcal{R}(x,w) = 1$, where the verifier sends its coins in the clear, and the prover sends commitments to its answers. At the end of the protocol, the prover and verifier use a zero-knowledge proof to prove that the verifier would accept the values in the commitments. The key point is that this zero knowledge proof is for a "smaller" statement, because the verifier of Theorem 1 is quite efficient. This last step uses a zero-knowledge proof with communication that is linear in the size of the verification circuit [CD97, IKOS07]. See [GKR15] for details. □

Interactive Proofs of Proximity. Interactive proofs of proximity [RVW13] are interactive proof systems in which the verifier runs in *sub-linear* time. Soundness is relaxed, and only guarantees that the verifier will reject inputs that are ϵ*-far* from the language (in fractional Hamming distance). We construct *constant-round* IPPs (with a polynomial-time honest prover) for languages computable in polynomial-time and bounded-polynomial space. [RVW13] construct IPPs for every language that can be computed by (uniform) bounded-depth circuits, where the round complexity grows with the circuit depth. See Section 1.3 for further discussion of related work.

Theorem 3. *Fix any sufficiently small constant $\sigma \in (0,1)$. Let $\mathcal{L}$ be a language that is computable in* $\mathsf{poly}(n)$*-time and $O(n^{\sigma})$-space, Then $\mathcal{L}$ has a constant-round ϵ-*IPP *for $\epsilon = n^{-1/2}$ with perfect completeness and soundness 1/2. The query and communication complexities, as well as the verifier's running time, are $n^{1/2+O(\sigma)}$. The (honest) prover runs in time* $\mathsf{poly}(n)$*.*

Proof Sketch. The IPP protocol of [RVW13] for (log-space uniform) bounded-depth circuits proceeds in two main steps. In the first step, the GKR protocol [GKR15] is run in parallel roughly $\sqrt{n}$ times to obtain a similar number of claims on the low degree extension of the input. Then, a special purpose sub-linear time protocol, called the PVAL protocol, is run to verify these claims. The GKR step in [RVW13] can be replaced with the protocol of Theorem 1. This uses the fact that, similarly to the GKR protocol, our verifier runs in sub-linear time given access to the input's low degree extension. In addition, the PVAL protocol of [RVW13] allows for a tradeoff between the number of rounds and communication, and in particular can be implemented in a constant number of rounds (with $n^{\frac{1}{2}+\varepsilon}$ communication for any constant $\varepsilon > 0$). See [RVW13] for further details. □

Delegating Uniform Bounded-Depth Computations. The interactive proof of [GKR15] applies to Logspace-uniform circuits of bounded depth. Theorem 1 can be used to extend their results to a richer notion of uniformity, giving interactive proofs for polynomial-time bounded-polynomial space uniform circuits of bounded depth.

Theorem 4. *Fix any sufficiently small constant $\sigma \in (0,1)$. Let $\mathcal{L}$ be a language computable by a $\mathsf{poly}(n)$-time $O(n^\sigma)$-space uniform ensemble of circuits of depth $D = D(n)$ and size $\mathsf{poly}(n)$. There is a public-coin interactive proof system for $\mathcal{L}$ with perfect completeness, constant soundness, and communication $\mathsf{poly}(n^\sigma, D)$. The number of rounds is $O(D \cdot \log n)$. The (honest) prover runs in time $\mathsf{poly}(n)$, and the verifier runs in time $n \cdot \mathsf{poly}(D, \log(n))$.*

Proof Sketch. The GKR protocol utilizes a "bare-bones" protocol for delegating bounded depth computations, where the verifier is assumed to have access to low-degree functions that specify the circuit's structure. This protocol does not assume *any* uniformity from the circuit. Building on the bare-bones protocol, they construct interactive proofs for Logspace-uniform circuits by "delegating" the computation of the functions that specify the circuit to the untrusted prover (this uses a separate interactive proof for delegating Logspace computations). For a polynomial-time bounded-polynomial space uniform circuit family, the (low-degree extensions of the) circuit-specifying functions can be computed in (uniform) polynomial time and bounded-polynomial space. Thus, we can also build on the bare-bones protocol, and use Theorem 1 to delegate the computation of the circuit-specifying functions to the untrusted prover. □

Delegating Randomized Computations. We also obtain doubly-efficient interactive proofs for bounded-space *randomized* computations (as opposed to deterministic computations) as a direct corollary of our main result. This extension is based on Nisan's [Nis92] pseudorandom generators (PRG) against bounded-space distinguishers. We note that this is in contrast to prior constructions of doubly efficient interactive-proofs (in particular [GKR08]), which apply only to deterministic computations.

Recall that the PRG of [Nis92] (see also [Gol08, Theorem 8.21]) implies that every randomized Turing machine M running in time T and space S can be simulated by a time $O(T)$ and space $O(S \cdot \log(T))$ randomized Turing Machine M' that uses only $O(S \cdot \log(T))$ random bits (up to some small constant error). An interactive proof for M' can be constructed by having the verifier first generate a random string r of length $O(S \cdot \log(T))$, and then send r to the prover. The two parties then run the interactive proof of Theorem 1 for the *deterministic* computation of M' when its random string is fixed to r.

1.3 Related Work and Related Models

A beautiful line of research, starting with the work of Babai, Fortnow, Lund and Szegedy [BFLS91], has focused on "PCP-like" proof systems with very efficient verifiers. In this model, the verifier has query access to a long *fixed* proof string, and can achieve verification time that is poly-logarithmic in the length of a general deterministic or non-deterministic computation ([BFLS91] assumes that the input is in an error-correcting code format, for general inputs verification requires quasi-linear time for encoding the input). In this work, and in its more efficient descendants [PS94, BGH+06, DR06, Din07], the PCP proof system's soundness requires that the proof string be *fixed*. While the proof might be wrong, it is static and does not change with the verifier's queries to it. If one thinks of delegating computation to an untrusted server over a network, it is not immediately clear how to leverage these results: the verifier needs to either "possess" the entire PCP proof string, which is longer than the entire computation (though only a few of its bits are eventually read), or to somehow have a guarantee that the prover/delegatee cannot adaptively change the PCP proof string as the verifier queries its bits. In contrast, our work focuses on the interactive proof setting,

where a cheating prover can be arbitrarily adaptive in its answers to the verifier's queries. As a consequence, our results are more amenable to delegating computation over a network.

A different model of interactive proof systems was studied by [EKR04, RVW13]. They consider sublinear-time verification with a natural approximate notion of soundness.[3] In Theorem 1, the verifier runs in nearly-linear time (i.e. verification is more expensive), but soundness holds for any input not in the language. In fact, Theorem 1 can also be used to construct new IPPs for bounded space computations, and these IPPs have the additional advantage of constant round complexity (the round complexity in [RVW13] grows with the circuit depth). Prior to our work, constant-round IPPs were only known for context-free languages, read-once branching programs [GGR15] and other specific languages [RVW13, GR15], or with only computational soundness (and under cryptographic assumptions) [KR15]. See Section 1.2 for further details.

Relaxing soundness to hold only against polynomial-time cheating provers, and assuming the existence of collision-resistant hash functions, Kilian [Kil92] constructed a 4-message argument system for any language in NP. The communication complexity is only polynomial in the security parameter (and logarithmic in the computation size). His framework can also be used for any language in P (regardless of its space complexity), and gives verifier runtime that is linear in the input length (and polynomial in the security parameter). We emphasize that an argument system achieves only computational soundness (soundness with respect to a computationally bounded dishonest prover). Our work focuses on the interactive proof setting, where soundness needs to hold against *any* cheating prover, and we make no cryptographic assumptions.

A line of works has attempted to reduce the interaction in argument systems even further. Micali [Mic94] shows that the interaction can be reduced in the random oracle model. This gives "non-interactive" proofs for the correctness of general computations (in the random oracle model). Kalai, Raz and Rothblum [KRR13, KRR14] constructed 2-message arguments for P, assuming quasi-polynomially hard PIR (or quasi-polynomially hard fully-homomorphic encryption). The assumption was more recently reduced to standard PIR by Brakerski *et al.* [BHK17]. The same work [BHK17] also constructs *computationally sound* batch verification protocols for NP.

Other works [Gro10, Lip12, BCCT12, DFH12, GLR11, BCCT13, GGPR13, BCI+13] show how to achieve 2-message arguments for NP under non-falsifiable assumptions (which are, to an extent, necessary [GW11]) or in the preprocessing model [GGP10, CKV10, AIK10, PRV12]. Other works obtain 2-message arguments for P assuming indistinguishability obfuscation [KLW15, BGL+15, CHJV15] or falsifiable assumptions on multi-linear maps [PR17].

Verifiable Delegation of Computation. Many of these works (including ours) are motivated by the application of delegating computations to an untrusted server in the setting of cloud computing. Indeed, some of these protocols and the ideas that underly them have been used in a burgeoning literature implementing secure systems for delegating computation. See the survey by Walfish and Blumberg [WB15] and the references therein for a more complete overview of the literature on verifiable delegation of computation and these more recent implementations.

Interactive Oracle Proofs. A model called *interactive oracle proofs* (IOPs), which is equivalent to our notion of probabilistically checkable interactive proofs (PCIP), was very recently introduced

[3] [RVW13] define *interactive proofs of proximity* (IPPs), where soundness is relaxed, and the verifier only needs to reject inputs that are *far* from the language. They construct sublinear-time IPPs for every language that can be computed by (uniform) bounded-depth circuits.

in an independent work of Ben Sasson *et al.* [BCS16] and studied also in [BCGV16] and [BCG+16]. [BCS16] define IOPs and study a compiler from public-coin IOPs to non-interactive proofs in the random oracle model. [BCGV16] define "duplex PCPs", which are 2-round IOPs, and construct duplex PCPs for NP that provide perfect zero-knowledge, with quasilinear-size proof length and constant query complexity. [BCG+16] continue the study of IOPs, and (among other contributions) construct linear-size IOPs for circuit-SAT. [BCG+16] also study IOPs of proximity which are equivalent to our notion of PCIPs of proximity (see Definition 4.9 below). We note that the "IOP composition" tool in [BCG+16], which is used to reduce the verifier's query complexity, is reminiscent of our "query reduction" transformation for PCIPs.

Followup Work. The recent work [RRR18] gives a batch verification protocol for UP that is more efficient than the one presented here. Specifically, the [RRR18] protocol avoids an additive $\Omega(k)$ factor that appears in our batch verification protocol. We remark however that we do not know how to use the [RRR18] to construct better interactive proofs for bounded space computations than the ones constructed here.

Other Related Notions. Batch verification of PCPPs (called *simultaneous* PCPP*s* therein) was used by Meir [Mei16, section 5.1] in his construction of short combinatorial PCPs.

A related problem to that of batch verification of NP statements, due to Harnik and Naor [HN10] (see also [Dru15] and references therein), is that of *AND-instance compression*. Here, we are given k statements and we wish to *efficiently* compress them so that the compressed string retains the information of whether all k statements hold. Strong infeasiblity results for AND-instance compression for NP [Dru15] are known. Our notion of batch verification for NP (or UP statements) can be thought of as a type of compression for *witnesses* rather than instances. Furthermore, our notion of batch verification is different from instance compression in that: (1) we have access to an (untrusted) prover who can help us compress, and (2) the compression is via an *interactive* protocol.

Another related notion is that of *Probabilistically Checkable Debate Systems*. Probabilistically Checkable Debate Systems (PCDS), introduced by Condon *et al.* [CFLS95] and recently studied by Drucker [Dru11], are protocols between a verifier and two competing provers, where one prover tries to prove that the input is in the language and other prover tries to prove that it is not. This is similar to refereed games [FK97], but here an additional catch is that the verifier is required to only read a few bits from the transcript in order to reach its verdict. PCDSes are loosely related to our notion of PCIP: in both proof-systems the verifier reads a few bits of the transcript. In contrast to PCDSes however, in PCIPs there is only one prover.

2 Overview of The Construction

An Iterative Construction. Assume we have a "base" interactive proof for verifying Turing Machine computations, where the Turing machine computations run in time T and space S. Our approach is to build on this protocol to construct an "augmented" interactive proof for verifying "longer" computations that run in time $(k \cdot T)$ and space S, where k is an integer (much) larger than 1. We use this augmentation step iteratively, starting with trivial interactive proofs for short computations, and gradually obtaining increasingly powerful interactive proofs for longer and longer computations.

We proceed with a discussion of the augmentation step. We begin with a base protocol, where prover and verifier agree on a (deterministic) Turing Machine $\mathcal{M}$, an input $x \in \{0,1\}^n$, and two configurations $u, v \in \{0,1\}^S$ (a configuration includes the machine's internal state, the contents of all memory tapes, and the position of the heads). The prover's claim is that after running the machine $\mathcal{M}$ on input x, starting at configuration u and proceeding for T steps, the resulting configuration is v. We denote this claim by:

$$(\mathcal{M}, x) : u \overset{T}{\rightsquigarrow} v.$$

We augment the base protocol, using it to design a new protocol for verifying longer computations running in time $(k \cdot T)$, i.e. "augmented claims" of the form:

$$(\mathcal{M}, x) : u \overset{(k \cdot T)}{\rightsquigarrow} v.$$

Consider an augmented claim, where u is the initial configuration, and v is the alleged configuration after $(k \cdot T)$ steps. The prover's first message in the augmented protocol is $(k-1)$ alleged intermediate configurations

$$(\tilde{w}_T, \tilde{w}_{2T}, \ldots, \tilde{w}_{(k-1)\cdot T}),$$

where $\tilde{w}_t$ is the alleged configuration of the machine $\mathcal{M}$ after t steps (with initial configuration u and on input x).[4] In particular, $\tilde{w}_0 = u$ and $\tilde{w}_{(k \cdot T)} = v$. The $(k-1)$ intermediate configurations sent by the prover specify k "base claims" about T-step computations: For each $j \in [k]$, the prover claims that the machine $\mathcal{M}$, starting from configuration $\tilde{w}_{(j-1)\cdot T}$, and running for T steps, reaches configuration $\tilde{w}_{j \cdot T}$. That is,

$$\forall j \in [k] : (\mathcal{M}, x) : \tilde{w}_{(j-1)\cdot T} \overset{T}{\rightsquigarrow} \tilde{w}_{j \cdot T}$$

The verifier should accept if and only if all k of these base claims are true.

In a naive augmentation, the verifier runs the base protocol k times to verify all k of the base claims. This increases the communication and verification time by a multiplicative factor of k. While the resulting augmented protocol can be used to verify computations that are k times longer then the base protocol, it is also k times more expensive, so we have not made any real progress.

Another naive option is picking just one (or several) of the base claims, and verifying only them. This is less expensive in communication and verification time, but the soundness error grows prohibitively. In particular, suppose that the prover is cheating, and the computation path of length $k \cdot T$ that starts at $\tilde{w}_0$ does not end at $\tilde{w}_{k \cdot T}$. The cheating prover can still generate a sequence $(\tilde{w}_T, \tilde{w}_{2T}, \ldots, \tilde{w}_{(k-1)\cdot T})$ where all but one of the base claims are true. For example, the cheating prover could pick $j^* \in [k]$, set the configurations $(\tilde{w}_T, \ldots, \tilde{w}_{(j^*-1)\cdot T})$ to be the appropriate configurations on a path of length $((j^*-1) \cdot T)$ that starts at $\tilde{w}_0$ (and ends at $\tilde{w}_{(j^*-1)\cdot T}$), and set the configurations $(\tilde{w}_{j^* \cdot T}, \ldots, \tilde{w}_{(k-1)\cdot T})$ to be the appropriate configurations on a path of length $((k-j^*) \cdot T)$ that starts at $\tilde{w}_{j^* \cdot T}$ and ends at $\tilde{w}_{k \cdot T}$. Now all of the base claims are true, except for the $j^{*\text{th}}$ (where there is no path of length T from $\tilde{w}_{(j^*-1)\cdot T}$ to $\tilde{w}_{j^* \cdot T}$). Unless the verifier checks all (or very many) of the base claims, it will fail to detect any cheating.

What we seek is a protocol for verifying the k base claims, but with communication and verification time that is much smaller than running the base protocol k times, and with soundness

[4] Here and throughout this work we use tildes to denote potentially-corrupted strings that the verifier receives from an untrusted prover.

error that is not much larger than that of the base protocol. Also, the number of rounds should not grow too much (so that we can get interactive proofs with a small number of rounds), and the complexity of the (honest) prover should only grow by a factor of roughly k (so we can get a doubly-efficient proof system). We refer to this goal as *"batch verification of interactive proofs"*. We emphasize that, as described above, it is crucial that if even just one of the claims is false, the verifier should still reject.

One of our contributions is an efficient *Batch Verification Theorem* for a certain class of interactive proofs (so-called *unambiguous* interactive proofs, see below). The Batch Verification Theorem enables an augmentation step, where the cost of the augmented protocol is only moderately larger than the cost of the base protocol. Using this augmentation step, we iteratively construct efficient interactive proofs for long computations, and obtain the result of Theorem 1.

The remainder of this section is devoted to an overview of key ideas underlying the Batch Verification Theorem for unambiguous interactive proofs. We begin by considering the more modest goal of batching the verification of UP statements (NP statements that have at most one witness), which gives a taste of our new ideas and techniques. We then briefly discuss the additional challenges in batching (unambiguous) interactive proof systems.

2.1 A Primer: Batch Verification of UP Statements

To illustrate some of the ideas behind the batch verification theorem, we consider the simpler challenge of designing an interactive proof system for batch verification of UP *statements*. Recall that the complexity class UP (unambiguous non-deterministic polynomial-time) is the subset of NP problems where the non-deterministic Turing Machine has at most one accepting path. That is, for a language $\mathcal{L} \in \mathsf{UP}$, and an input $x \in \mathcal{L}$, there is *exactly* one witness to x's membership (and for $x \notin \mathcal{L}$ there are no witnesses). Batch verification of general NP statements is a fascinating open question.[5]

Consider a UP language $\mathcal{L}$, with witnesses of length $m = m(|x|)$. Our goal is to design an *interactive proof* $(\mathcal{P}^{\mathsf{IP}}, \mathcal{V}^{\mathsf{IP}})$ where, given k inputs $x_1, \ldots, x_k$, the verifier accepts only if $\forall j \in [k], x_j \in \mathcal{L}$ (otherwise the verifier rejects w.h.p.). We also want the prover strategy to be *efficient*: the (honest) prover should run in polynomial time given witnesses to the inputs' membership in $\mathcal{L}$. A sound but naive protocol is for the prover $\mathcal{P}^{\mathsf{IP}}$ to send all k witnesses $w_1, \ldots, w_k$, and for the verifier $\mathcal{V}^{\mathsf{IP}}$ to verify every pair (x_j, w_j). This protocol is sound (indeed, it is a UP proof system with soundness error 0), but it is very expensive, requiring communication $(k \cdot m)$ (and k witness verifications). Our goal is to *batch* the verification of these k UP statements via an interactive proof with communication (and verification time) that is (much) smaller than $(k \cdot m)$. In what follows we show an interactive proof with communication $(\mathsf{polylog}(k, n) \cdot (k + \mathsf{poly}(m)))$.

Theorem 5 (Batch Verification Theorem for UP)**.** *Let $\mathcal{L}$ be a language in UP with witnesses of length $m = m(n) = \mathsf{poly}(n)$, and let $k = k(n) \geq 1$ be an ensemble of integers. There is an interactive proof that, on input $(x_1, \ldots, x_k) \in \{0,1\}^{k \cdot n}$, verifies that $\forall j \in [k], x_j \in \mathcal{L}$, with perfect completeness and soundness $1/2$.*

The communication complexity is $(\mathsf{polylog}(k, n) \cdot (k + \mathsf{poly}(m)))$. The number of rounds is $\mathsf{polylog}(k)$. The running time of the verifier is $(\mathsf{polylog}(k, n) \cdot ((k \cdot n) + \mathsf{poly}(m)))$. The (honest)

[5] We note that we do not see a way to deduce a similar theorem for general NP statements by applying the Valiant-Vazirani randomized reduction from NP to UP [VV86].

prover, given witnesses $(w_1, \ldots, w_k) \in \{0,1\}^{k \cdot m}$ *for the inputs' membership in* $\mathcal{L}$*, runs in time* $\mathsf{poly}(k, n)$.

The remainder of this section is devoted to a proof sketch for Theorem 5. The full proof follows along similar lines to the batch verification theorem for interactive proofs (see Section 9). Before proceeding, we make two remarks:

Remark 2.1 (Batch Verification for Different Languages)**.** *Theorem 5 holds even w.r.t.* k different *langauges* $\mathcal{L}_1, \ldots, \mathcal{L}_k$*, all in* UP*. The interactive proof verifies that* $\forall j \in [k], x_j \in \mathcal{L}_j$.

Remark 2.2 (Relationship to low-communication interactive proofs)**.** *There are significant barriers to the construction of low-communication interactive proofs for general* NP *languages, and these extend to* UP*. Goldreich and Håstad [GH98] show that if a language* $\mathcal{L}$ *has an interactive proof-system, where the total communication is bounded by* c *bits, then* $\mathcal{L}$ *can be decided in time that depends only exponentially on* c*. Goldreich, Vadhan and Wigderson [GVW02] show that if* $\mathcal{L}$ *has an interactive proof where the prover sends only* b *bits to the verifier (regardless of the length of communication from the verifier to the prover), then there is a* constant-round *interactive proof for* $\mathcal{L}$*'s complement, with communication and verification time that depends only exponentially on* b*. With these results in mind, consider unique-SAT: the promise problem*[6] *where YES instances are formulae with a single satisfying assignment, and NO instances have 0 satisfying assignments. It would be surprising to construct an interactive proof for unique-SAT where the prover-to-verifier communication is significantly shorter than the witness length (say a small polynomial), as this would place co-SAT in the sub-exponential analogue of* AM *(an interactive proof with short total communication would be even more surprising, placing co-SAT in sub-exponential time and contradicting the exponential-time hypothesis).*[7] *However, this obstacle should not (and does not) deter us from* batching *the verification of* UP *statements. We seek an interactive proof with communication* b *for the language:*

$$\mathcal{L}^{\otimes k} = \{(x_1, \ldots, x_k) : \forall j \in [k], x_i \in \mathcal{L}\}.$$

This is an UP *language, with witnesses of length* $(k \cdot m)$ *(assuming* $\mathcal{L}$ *has witness length* m*). However, whatever the complexity of* $\mathcal{L}$*, the language* $\mathcal{L}^{\otimes k}$ *has additional structure. In particular,* $\mathcal{L}^{\otimes k}$ *and its complement can be decided in time* $(k \cdot 2^m \cdot \mathsf{poly}(n))$ *by doing a brute-force search to determine whether every one of the* k *statements has a witness. Thus, the results of [GH98, GVW02] do not present a barrier to batch verification of* $\mathcal{L}$ *using an interactive proof with communication* $b = \Omega(\log(k) + m + \log(n))$.

Remark 2.3. *As mentioned above, the recent work of [RRR18] gives an improved batch verification protocol for* UP *that avoids the additive* k *factor in the communication complexity in Theorem 5.*

A Tantalizing (but Flawed) Protocol. We begin by considering a (flawed) attempt to use PCPs to design a sound batch verification protocol. For this, assume that the language $\mathcal{L} \in \mathsf{UP}$ has a PCP proof system with proofs of length $a = a(n) = \mathsf{poly}(m)$, and a verifier $\mathcal{V}^{\mathsf{PCP}}$ who makes at most $q = q(n) = O(\mathsf{polylog}(a(n)))$ queries. We assume that the PCP verifier is non-adaptive, and its queries depend only on its random coins (as is the case for standard constructions). As $\mathcal{L} \in \mathsf{UP}$

[6] Note that the results of [GVW02] hold also for promise problems (rather than just language membership (see discussion in [GVW02, Section 2]).

[7] The [GH98, GVW02] results would imply upper bounds for the complement of unique-SAT. Upper bounds for co-SAT follow by the Valiant-Vazirani reduction [VV86].

we can assume that for each $x \in \mathcal{L}$, there is a *unique* PCP $\alpha \in \{0,1\}^a$ for x's membership in $\mathcal{L}$ that makes the verifier accepts α (on input x) with probability 1.[8] We note that this is the main reason we need $\mathcal{L}$ to be a UP language (rather than any language in NP). Using a PCP with the above properties, we design an interactive proof $(\mathcal{P}^{\mathsf{IP}}, \mathcal{V}^{\mathsf{IP}})$ for verifying that $\forall j \in [k], x_j \in \mathcal{L}$.

Consider the following tantalizing (but insecure) protocol. The verifier $\mathcal{V}^{\mathsf{IP}}$ runs $\mathcal{V}^{\mathsf{PCP}}$ to generate k sets of PCP queries for verifying each of the k statements. Since we assume that $\mathcal{V}^{\mathsf{PCP}}$ is non-adaptive, $\mathcal{V}^{\mathsf{IP}}$ can use the same random coins for verifying all k statements, and they will issue the same set $S \subset [a]$ of queries. $\mathcal{V}^{\mathsf{IP}}$ sends the query-set S to the untrusted prover, receives answers for each of the k PCPs, and accepts if and only if for every $j \in [k]$, the answers provided for the j^{th} PCP make $\mathcal{V}^{\mathsf{PCP}}$ accept on input x_j. This requires only roughly $O(k \cdot q) = O(k \cdot \mathsf{polylog}(a))$ communication, but it does not guarantee *any* soundness. The problem is that a cheating prover in the interactive proof setting is completely adaptive, and can tailor its responses to $\mathcal{V}^{\mathsf{PCP}}$'s queries in an arbitrary manner. Even if $x_{j^*} \notin \mathcal{L}$, *after* the cheating interactive-proof prover sees the queries made by $\mathcal{V}^{\mathsf{PCP}}$, it can tailor answers that make $\mathcal{V}^{\mathsf{PCP}}$ accept. The PCP's soundness is only guaranteed if the entire proof string is fixed in advance, *before* the PCP verifier's queries are made.

Towards Sound Batch Verification. Building on the "tantalizing protocol" , we now present our first attempt for a sound batch verification protocol. We assume that the honest prover $\mathcal{P}^{\mathsf{IP}}$ is given as input k PCP proofs, where $\alpha_j \in \{0,1\}^a$ is a PCP for the j^{th} statement x_j. The protocol proceeds as follows:

1. $\mathcal{P}^{\mathsf{IP}}$ constructs a $k \times a$ matrix A whose rows are the PCP proofs for the k statements:

$$A = \begin{pmatrix} & \alpha_1 & \\ & \alpha_2 & \\ & \cdots & \\ & \alpha_k & \end{pmatrix}.$$

   $\mathcal{P}^{\mathsf{IP}}$ computes the parity of A's columns, and sends these parities to the verifier. We view this vector of parities (one per column of A) as a "checksum" $chksum = \bigoplus_{j \in [k]} \alpha_j$.

2. $\mathcal{V}^{\mathsf{IP}}$ receives a vector $\widetilde{chksum} \in \{0,1\}^a$. It proceeds to choose a single set of random coins for the PCP verifier $\mathcal{V}^{\mathsf{PCP}}$. These coins specify a set $S \subseteq [a]$ of q queries to the k (alleged) PCPs, and $\mathcal{V}^{\mathsf{IP}}$ sends S to $\mathcal{P}^{\mathsf{IP}}$. (Recall that we assume that the PCP verifier's queries are non-adaptive, and depend only on its random coins).

3. $\mathcal{P}^{\mathsf{IP}}$ receives the set S of coordinates, and for every $j \in [k]$ it sends back the values of the j^{th} PCP (the j^{th} row), restricted to the q entries in S. We view the answers for the j^{th} row as an assignment $\phi_j : S \to \{0,1\}$.

4. The verifier $\mathcal{V}^{\mathsf{IP}}$ runs two tests (and accepts only if they both pass):

[8] In fact, we can allow more than one PCP string that makes the verifier accept with probability 1. All that we require is that there exist a unique such PCP string α, where given x and a candidate PCP $\alpha' \in \{0,1\}^a$, we can test in polynomial time whether $\alpha = \alpha'$. This property is satisfied by standard PCP constructions (applied to UP statements), and it comes for free when there is a single fully-correct α (as above), so long as q is a constant. We mention that Goldreich and Sudan [GS06, Definition 5.6] studied a stronger notion of PCPs, called strong PCPs, in which the rejection probability of the PCP verifier needs to be proportional to the distance of the given PCP from the prescribed PCP.

(a) **PCP Check.** For every $j \in [k]$, the prover's PCP answers $\{\phi_j(\xi)\}_{\xi \in S}$ make $\mathcal{V}^{\mathsf{PCP}}$ accept the input x_j (the same random string, chosen above, is used for all k PCP verifications).

(b) **Consistency Check.** For every query $\xi \in S$, the ξ^{th} bit of $\widetilde{chksum}$ indeed equals the parity of the values claimed for the ξ^{th} column of A. That is:

$$\forall \xi \in S : \widetilde{chksum}[\xi] = \bigoplus_{j \in [k]} \phi_j(\xi).$$

This batch verification protocol is quite efficient: the communication complexity is only $(a + O(k \cdot q))$ bits, a considerable savings over the naive sound protocol that required $(k \cdot a)$ bits. The verifier $\mathcal{V}^{\mathsf{IP}}$ runs in time $O(a + k \cdot |\mathcal{V}^{\mathsf{PCP}}|)$, where $|\mathcal{V}^{\mathsf{PCP}}|$ is the running time of the PCP verifier. The (honest) prover's running time (given the k PCPs) is $O(k \cdot a)$, and there are 3 messages exchanged.

Soundness for Single-Deviations. The question, of course, is whether the protocol is sound (completeness follows by construction). Unfortunately, the protocol is not sound in general.[9] However, it *is* sound against an interesting class of cheating provers, which we call *single-deviation provers*. For this, we focus on proving soundness when there is only a single $j^* \in [k]$ s.t. $x_{j^*} \notin \mathcal{L}$. In Step 3 (answering the PCP queries), we restrict the cheating prover $\widetilde{\mathcal{P}}$ as follows. For every $j \neq j^*$, $\widetilde{\mathcal{P}}$ knows the *unique* "correct" PCP $\alpha_j \in \{0,1\}^a$ (see above) that makes the verifier $\mathcal{V}^{\mathsf{PCP}}$ accept the input x_j with probability 1 (note that α_j, being the unique correct PCP, is fixed in advance before the protocol begins). In Step 3 of the protocol, $\widetilde{\mathcal{P}}$ answers all queries to the j^{th} PCP (for $j \neq j^*$) according to α_j. We emphasize that $\widetilde{\mathcal{P}}$ is unrestricted in Step 1; it can send an arbitrary $\widetilde{chksum}$, and it can send arbitrary and adaptive answers to the $j^{*\text{th}}$ PCP in Step 3 (after seeing the query set S). In particular, the tantalizing protocol is completely insecure even against single-deviation cheating provers.

We show that the batch verification protocol described above *is* sound against single-deviation cheating provers. Suppose that a cheating single-deviation prover $\widetilde{\mathcal{P}}$ makes the verifier accept with probability ϵ. We use $\widetilde{\mathcal{P}}$ to construct a *fixed* proof $\tilde{\alpha}_{j^*}$ that makes the PCP verifier accept the input $x_{j^*} \notin \mathcal{L}$ with probability ϵ, and conclude that the interactive proof protocol is sound. We derive $\tilde{\alpha}_{j^*}$ from the checksum value $\widetilde{chksum}$ sent by $\widetilde{\mathcal{P}}$ in Step 1 (w.l.o.g. the cheating prover is deterministic and its first message is fixed):

$$\tilde{\alpha}_{j^*} = \widetilde{chksum} \oplus \left(\bigoplus_{j \neq j^*} \alpha_j \right)$$

We claim that on input x_{j^*} the PCP verifier $\mathcal{V}^{\mathsf{PCP}}$ will accept $\tilde{\alpha}_{j^*}$ with probability ϵ. To see this, recall that $\widetilde{\mathcal{P}}$ answers all queries to rows $j \neq j^*$ according to α_j. Whenever $\widetilde{\mathcal{P}}$ makes $\mathcal{V}^{\mathsf{IP}}$ accept, it must pass the consistency check in Step 4b, and thus it must answer the queries to the $j^{*\text{th}}$ PCP according to $\tilde{\alpha}_{j^*}$. Since it also needs to pass the PCP check in Step 4a, we conclude that whenever $\widetilde{\mathcal{P}}$ makes $\mathcal{V}^{\mathsf{IP}}$ accept, it must also be the case that the PCP answers $(\tilde{\alpha}_{j^*}|_S)$ make the PCP verifier $\mathcal{V}^{\mathsf{PCP}}$ accept on input x_{j^*}.

[9] Consider inputs $x_1, \ldots, x_{k-2} \in \mathcal{L}$ and $x_{k-1} = x_k = x^* \notin \mathcal{L}$. Consider a cheating prover that generates the correct PCPs $\alpha_1, \ldots, \alpha_{k-2}$ for $x_1, \ldots, x_{k-2}$, and sends $\widetilde{chksum} = \bigoplus_{j \in [k-2]} \alpha_j$ to the verifier (i.e., the checksum excludes the last two inputs). Once the verifier sends PCP queries S, the prover answers honestly on all but the last two rows. For the latter two rows, it finds some assignment $\tilde{\phi} : S \to \{0,1\}$ that satisfies the PCP verifier w.r.t input x^* and queries S (this is easy to do given S), and sends $\tilde{\phi}$ as the answer to the PCP queries for rows $k-1$ and k. The two $\tilde{\phi}$'s cancel out and so the consistency check passes and the verifier accepts.

Implicit Commitments and Soundness for d Deviations. Reflecting on the soundness of the above batch verification protocol, observe that $\widetilde{chksum}$ *implicitly* commits a single-deviation prover to the PCP string $\tilde{\alpha}_{j^*}$ for $x_{j^*} \notin \mathcal{L}$. Once the prover is committed, soundness of the protocol naturally follows from the soundness of the PCP. Of course, $\widetilde{chksum}$ is much too short to include an *explicit* commitment to the k PCP strings (for the k inputs x_j). Thus, we should not expect soundness against general provers (indeed it is not clear how to leverage the PCP's soundness against general adaptive provers). Nevertheless, it is not hard to generalize the above batch verification protocol to handle d deviations as long as d is not too large.

To extend soundness, in Step 1 of the protocol, we ask the prover to send a "more robust" $O(d \cdot \log k)$-bit checksum for each column of the matrix A, where this checksum has the property that for every $y \in \{0,1\}^k$ and $z \in \{0,1\}^{O(d \cdot \log k)}$, there is at most one $y' \in \{0,1\}^k$ at Hamming distance d or less from y whose checksum equals z (including y itself). We can construct such a checksum using standard techniques from the error-correcting code literature (see Proposition 6.6 for details). Putting together these checksums (one per column of A), we get a matrix $chksum \in \{0,1\}^{O(d \cdot \log k) \times a}$, which $\mathcal{P}^{\mathsf{IP}}$ sends to $\mathcal{V}^{\mathsf{IP}}$. The verifier $\mathcal{V}^{\mathsf{IP}}$ receives a potentially-corrupted checksum $\widetilde{chksum} \in \{0,1\}^{O(d \cdot \log k) \times a}$, and in Step 4b, it checks that the PCP answers are consistent with this "more robust" checksum. The protocol is unchanged otherwise. Note that this increases the communication to $O((d \cdot \log k \cdot a) + (k \cdot q))$, which remains interesting so long as $d \ll k$.

The new checksum matrix $chksum$ is still not long enough to commit an arbitrary prover to k PCP strings. But intuitively it can implicitly commit a prover as long as it does not deviate on more than d rows. More formally, for every $j \neq j^*$,[10] the cheating prover $\widetilde{\mathcal{P}}$ knows the unique PCP α_j. After the verifier specifies the query set S in Step 2, a d-deviation prover $\widetilde{\mathcal{P}}$ (adaptively) chooses a set $J^* \subset [k]$ of d of the instances (or rows), and can provide arbitrary answers on queries to those d PCPs. The only restriction is that for every $j \notin J^*$, $\widetilde{\mathcal{P}}$ answers the queries to the j^{th} PCP according to the predetermined PCP α_j.

Similarly to the argument for single-deviation prover, it can be shown that the possibly corrupt checksum string $\widetilde{chksum} \in \{0,1\}^{O(d \cdot \log k) \times a}$ induces an *implicit* commitment to a PCP string $\tilde{\alpha}_{j^*}$ for x_{j^*} (the input that is not in $\mathcal{L}$). In fact, it induces commitments to all the k PCP strings. Of course, this argument only works because we restricted $\widetilde{\mathcal{P}}$ to d deviations.

Amplifying Deviations and a $\sqrt{k}$ Batch Verification. We described a batch verification protocol that is sound for d deviations. We will now show how to exploit it against a general cheating prover (even though it is not itself sound for such a prover). The key observation is that while even the more robust checksum does not directly induce a commitment to the $j^{*\text{th}}$ PCP, it does tie $\widetilde{\mathcal{P}}$'s hands in an important way. In answering PCP queries for the inputs $\{x_j\}_{j \neq j^*}$ *that are in the language*, $\widetilde{\mathcal{P}}$ is faced with two hard choices: It can provide answers that are mostly consistent with the correct PCPs (on all but d of the rows), but then soundness against d deviations implies that $\mathcal{V}^{\mathsf{IP}}$ will reject. Alternatively, if $\widetilde{\mathcal{P}}$ deviates on d or more rows, then it is sending answers that are inconsistent with the *unique* correct PCPs, and this is much easier for the verifier to detect (as we show next).

To obtain a sound batch verification protocol, we add an additional round of communication, where $\mathcal{V}^{\mathsf{IP}}$ picks $O(k/d)$ of the rows at random and asks $\widetilde{\mathcal{P}}$ to send those rows' PCPs (in their entirety). Since $\widetilde{\mathcal{P}}$ deviated from the unique correct PCPs on at least d rows, it is likely that there

[10] Recall that we assume that there is exactly one row j^* s.t. $x_{j^*} \notin \mathcal{L}$ (this assumption is for simplicity and without loss of generality, see Remark 2.4 below).

is some row j that $\mathcal{V}^{\mathsf{IP}}$ requested where $\widetilde{\mathcal{P}}$ has deviated. Either $\widetilde{\mathcal{P}}$ sends a PCP that is inconsistent with its past answers, or it is forced to send $\tilde{\alpha}_j$ that is not the unique correct PCP for $x_j \in \mathcal{L}$. In either case, $\mathcal{V}^{\mathsf{IP}}$ rejects.

The final check adds $O((k/d) \cdot a)$ communication bits (and verification time), and results in a sound protocol for batch verification of UP statements. Setting $d = \sqrt{k}$, we obtain a protocol with $\tilde{O}(\sqrt{k} \cdot a + k \cdot q)$ communication (compared with $(k \cdot a)$ for the naive protocol).

Here, we use the protocol that is secure against d-deviation provers as a "deviation amplification" protocol. We find it noteworthy that this deviation amplification forces $\widetilde{\mathcal{P}}$ to cheat (and get caught!) on inputs *that are in the language*. This is one of the key insights in constructing our batch verification protocols. Note that here we crucially use the property that for each $x \in \mathcal{L}$, there is a unique correct PCP α, and the verifier can check whether $\alpha' = \alpha$ in polynomial time. This is also why our soundness argument applies to UP statement, but does not extend to general NP statements.

Remark 2.4. *[Many inputs not in $\mathcal{L}$] We assumed throughout that there was only a single $j^* \in [k]$ for which $x_{j^*} \notin \mathcal{L}$. More generally, the protocol is sound for any number of inputs that are not in $\mathcal{L}$. Soundness for the general case is shown via a similar argument: if there are less than d inputs that are not in the language, then the above argument goes through in a very similar manner. If there are more than d inputs that are not in the language, then when $\mathcal{V}^{\mathsf{IP}}$ picks $O(k/d)$ statements and checks them explicitly, it will likely "catch" an input that is not in $\mathcal{L}$ and will reject, since the cheating prover cannot supply an accepting PCP for a false statement.*

Improving the Dependence on k. Finally, we turn our attention to improving the communication to $(\mathsf{polylog}(k) \cdot (a + k \cdot q))$, as claimed in Theorem 5. We begin with the d-deviation protocol. Recall that we can use this protocol to amplify deviations, forcing a cheating $\widetilde{\mathcal{P}}$ to send "incorrect" PCP values for at least d rows. As above, $\mathcal{V}^{\mathsf{IP}}$ chooses a random set $J_1 \subset [k]$ of size $O(k/d)$, and we know that with good probability over the choice of J_1, there is at least one row $j \in J_1$ for which either $x_j \notin \mathcal{L}$, or $x_j \in \mathcal{L}$, but $\widetilde{\mathcal{P}}$ sent incorrect PCP values: $\exists \xi \in S : \phi_j(\xi) \neq \alpha_j|_\xi$ (where α_j is the unique correct PCP for x_j). Above, $\mathcal{V}^{\mathsf{IP}}$ detected this by asking $\widetilde{\mathcal{P}}$ to send the correct PCP for every $j \in J_1$. This guaranteed soundness, but at a cost of $(|J_1| \cdot a)$ communication.

Observe, however, that once $\mathcal{V}^{\mathsf{IP}}$ picks J_1, we are in a familiar situation: we have a relatively large set J_1 of statements, and would like to detect whether $\widetilde{\mathcal{P}}$ is "cheating" on at least one of these statements, but without explicitly sending all $|J_1|$ witnesses. The natural approach is to recurse: use the deviation amplification protocol to amplify the number of deviations *within* J_1 to at least d rows, pick a smaller set $J_2 \subset J_1$ of size $O(|J_1|/d)$, and recurse again and again until we have a set J_{final} of size $O(d)$ and for some $j \in J_{\mathit{final}}$ we have $x_j \notin \mathcal{L}$ (or the prover deviated from the prescribed protocol PCP for j). At the "base" of this recursion, the prover can send explicit witnesses for each $j \in J_{\mathit{final}}$. Each run of the deviation amplification protocol only requires $O(d \cdot \log(k) \cdot a + k \cdot q)$ communication, so by setting $d = \log k$ we can get a recursion of depth $O(\log k)$ and a total communication cost of $(\mathsf{polylog}(k) \cdot (a + k \cdot q))$ (with $O(\log k)$ rounds). More generally, we could use different values of d to trade off the number of rounds for communication (and in particular to obtain constant-round protocols).

There is a subtlety in the argument outlined above. Namely, in the recursion, *the* UP *language has changed*. The statement we want to verify for each row $j \in J_1$ is that both: (i) The j^{th} input is in the language, i.e. $x_j \in \mathcal{L}$ (as before), and (ii) For the set S of PCP queries chosen by the verifier and each ϕ_j sent by the prover, the correct PCP α_j for x_j satisfies $\phi_j(\xi) = \alpha_j|_\xi$, for every

$\xi \in S$. These two conditions define a new language $\mathcal{L}_1$ over triplets (x, S, ϕ), and we want to verify that $\forall j \in J_1, (x_j, S, \phi_j) \in \mathcal{L}_1$. First, observe that if $\mathcal{L} \in \mathsf{UP}$ then also $\mathcal{L}_1 \in \mathsf{UP}$. Moreover, using algebraic PCP techniques, we can modify the PCP system for $\mathcal{L}$ to get a PCP system for $\mathcal{L}_1$ with only a small loss in the parameters. Using this modified PCP, we can indeed realize the recursion outlined above, and this yields the batch verification protocol of Theorem 5.

2.2 Batching Unambiguous Interactive Proofs

The iterative interactive proof construction of Theorem 1 is based on an efficient Batch Verification Theorem for interactive proofs, which builds on the ideas for batch UP verification outlined in Section 2.1. Towards this goal, we introduce interactive analogues of the building blocks used in the proof of the UP batch verification theorem. We discuss these new proof system notions, which may be of independent interest, in Section 2.2.1. In Section 2.2.2 we use them to give an overview of the batching theorem for interactive proofs, and give further details on the iterative construction of Theorem 1.

2.2.1 Unambiguous and Probabilistically Checkable Interactive Proofs

In the setting of UP-verification it was important that for inputs in the language we had a single PCP proof string that convinces the verifier. We introduce here an interactive analogue of these "unambiguous" proof strings, which we call *unambiguous interactive proofs*.

Unambiguous Interactive Proofs (UIPs). An *unambiguous* interactive proof system for a language $\mathcal{L}$ is specified by a deterministic (honest) prover $\mathcal{P}$, which we call the *prescribed prover*, and a verifier $\mathcal{V}$ (much like a classical interactive proof system). Suppose that a cheating prover $\widetilde{\mathcal{P}}$ follows the protocol in rounds $1, \ldots, i-1$, but "deviates" in round i, sending a message different from the prescribed message that $\mathcal{P}$ would have sent. In an unambiguous interactive proof, we require that for any round i where $\widetilde{\mathcal{P}}$ might first deviate, and for any history in rounds $1, \ldots, i-1$ (which is determined by $\mathcal{V}$'s coin tosses and by the prescribed prover strategy), if the prescribed prover would have sent message $\alpha^{(i)}$, but the cheating prover sends a message $\tilde{\alpha}^{(i)} \neq \alpha^{(i)}$, then the verifier will reject with high probability over its coin tosses in subsequent rounds. Note that this requirement also holds *for inputs that are in the language*, whereas the classical notion of an interactive proof does not make any restriction for such inputs. For inputs that are not in the language, the prescribed prover's first message is a special symbol that tells $\mathcal{V}$ to reject. In particular, if $x \notin \mathcal{L}$, but a cheating prover $\widetilde{\mathcal{P}}$ tries to convince $\mathcal{V}$ to accept, then $\widetilde{\mathcal{P}}$ needs to deviate from $\mathcal{P}$'s strategy in its first message, and the unambiguity property guarantees that w.h.p. $\mathcal{V}$ will reject. Thus, any unambiguous IP for $\mathcal{L}$ also guarantees the classical notion of soundness, and is also an interactive proof for $\mathcal{L}$. We note that UP proofs correspond to 1-message deterministic UIPs. See Definition 4.2 for a formal definition.

Remark 2.5. *It may be helpful to consider some examples of protocols that are unambiguous. A prominent example is the classical Sumcheck protocol [LFKN92]. There, in every round i, the (honest) prover sends the verifier a low-degree polynomial $P^{(i)}$, and the verifier checks the value of $P^{(i)}$ at a random point $\beta^{(i)}$ (we gloss over the details of this check). If a cheating prover sends a low-degree polynomial $\tilde{P}^{(i)} \neq P^{(i)}$, then w.h.p. over the verifier's choice of $\beta^{(i)}$ we have $\tilde{P}^{(i)}(\beta^{(i)}) \neq P^{(i)}(\beta^{(i)})$, and the verifier will end up rejecting. Building on this property of the sumcheck protocol,*

we note that the GKR interactive proof [GKR15] is also unambiguous. Another well-known example is the classical interactive proof for Graph Non-Isomorphism [GMW91]. On the other hand, zero-knowledge proofs are ambiguous *by design: The honest prover is randomized, and there are many messages that it can send that will end up making the verifier accept.*

Probabilistically Checkable Interactive Proofs (PCIPs). Analogously to the UP setting, where batch verification used the power of PCPs, we use a notion of probabilistic checking with low query complexity, but for *interactive* proof systems. That is, we use interactive proof systems where the verifier only reads a few bits of the transcript in checking the proof. We call these *probabilistically checkable interactive proofs* (PCIPs).

A PCIP for a language $\mathcal{L}$ is an interactive proof system, where the protocol is divided into two phases. In the *communication* phase, the prover and verifier interact for ℓ rounds and generate a transcript (as in a standard interactive proof). Restricting our attention to public-coin protocols, all that the verifier does in this phase is send random strings $\beta_1, \ldots, \beta_\ell$ (one in each of the ℓ rounds). In the *checking* phase, the verifier queries q bits of the messages sent by the prover and accepts or rejects. The verifier's running time in a PCIP is just the time for the checking phase (generating queries and deciding whether to accept). Thus, in a PCIP, the query complexity and the verifier's runtime can be much smaller than the transcript length. One can think of the prover and verifier as interactively generating a PCP (comprised of the prover's messages), which is then checked by the verifier. Indeed, a one-message PCIP is simply a PCP. For this overview, we assume that the queries do not depend on the input, only on the random coins chosen by the verifier in the communication phase. See Definitions 4.5 and 4.7 for formal definitions.

Remark 2.6 (Sublinear PCIPs)**.** *Another parameter that one may consider in the definition of* PCIP*s is the number of queries that the verifier makes into the input. Indeed, in the formalization of Definition 4.5 we bound the verifier's queries to the input, which facilitates more efficient batch verification of these objects. If the number of queries to the input is bounded, then building proof systems for general languages requires further relaxations. We can assume that the input is given in encoded form (as in the holographic proofs of [BFLS91]), or we can relax soundness to testing proximity to a language (as in* PCPP*s [BGH+06] and* IPP*s [RVW13]). In many places in this work we make the first relaxation and assume an encoded input. However, we find the second notion, which yields a notion of "*PCIP*s of proximity", to also be conceptually compelling and worthy of further exploration. For simplicity, throughout this overview, unless we explicitly note otherwise, we do not bound the verifier's input queries.*

Remark 2.7. *We compare* PCIP*s to other notions of interactive and probabilistically checkable proof systems. Kalai and Raz [KR08] study* interactive PCPs *(*IPCP*s), where the prover's first message to the verifier is a (long)* PCP *α, and the prover and verifier then run an interactive protocol to assist in checking the* PCP*. Their motivation was constructing succinct proofs for* NP *that are easy to verify and succinct zero-knowledge proofs. In an* IPCP*, the verifier reads only a few bits of α, but it reads* the entire transcript*. Indeed, one can think of these as interactive proof systems for verifying the validity of a* PCP*. In* PCIP*s, on the other hand, the verifier issues a bounded number of queries to the entire transcript.*

In an interactive proof of proximity *(*IPP*) [RVW13], the verifier runs in sublinear time, and can only make a sublinear number of queries to the input. The communication transcript with the prover is also of sublinear length, but the verifier reads the transcript in its entirety. In* PCIP*s, on*

the other hand, we do not restrict the transcript length to be sublinear, but the verifier makes only a small number of queries into this transcript (and on occasion we also bound the number of input queries, as in an IPP*, see Remark 2.6).*

Putting the two notions just discussed together, we define *unambiguous* PCIP*s*, which play a central role in the proof of the batch verification theorem. A subtlety that we mostly ignore in this overview is that full unambiguity cannot be obtained with small query complexity: If a cheating prover $\widetilde{\mathcal{P}}$ changes just one bit of the i^{th} message, and the verifier only makes a small number of queries to the message, this change will likely go unnoticed, and unambiguity is lost. There are several ways to reconcile these two notions, and the one most convenient for our purpose is to restrict the family of cheating provers such that every message sent by the cheating prover (as well as by the prescribed prover) is a codeword in a high-distance error-correcting code (the low-degree extension). We refer to this notion as unambiguous PCIP *w.r.t. encoded provers*. For a more complete discussion of the notions of interactive proofs we introduce here and their relations, as well as for their formal definitions, see Section 4.

2.2.2 Batch Verification using Unambiguous PCIPs

We are now ready to describe the batch verification protocol for unambiguous IPs. Given an unambiguous interactive proof $(\mathcal{P}, \mathcal{V})$ for a language $\mathcal{L}$, we obtain an interactive proof for verifying k instances of $\mathcal{L}$. This result is stated informally in Theorem 6 below.

Theorem 6 (Batch Verification Theorem for Unambiguous IP, Informal)**.** *Let $(\mathcal{P}, \mathcal{V})$ be an unambiguous interactive proof for language $\mathcal{L}$, with perfect completeness, soundness (or unambiguity) error ϵ, ℓ rounds, communication c, and prover and verifier runtimes $\mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}$. Let $0 < \tau \ll 1$ be a constant.*

Then, there is an batched *unambiguous interactive proof $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$ that, on input $(x_1, \ldots, x_k) \in \{0,1\}^{k \cdot n}$, verifies that $\forall j \in [k], x_j \in \mathcal{L}$, with perfect completeness and soundness (or unambiguity) error $O(\epsilon)$. The batched protocol $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$ has $O(\ell)$ rounds, communication complexity $(\mathsf{poly}(\ell, \mathcal{V}\mathsf{time}) \cdot k^{\tau} \cdot c)$, prover runtime $(\mathsf{poly}(\ell) \cdot k \cdot \mathcal{P}\mathsf{time})$ and verifier runtime $(\mathsf{poly}(\ell) \cdot k \cdot \mathcal{V}\mathsf{time})$.*

We note that an *inefficient* batch verification theorem for (general) interactive proofs also follows from the IP = PSPACE theorem. However, that protocol does not preserve the round complexity or prover-efficiency of the base protocols (and is thus not helpful for constructing interactive proofs with efficient provers or small round complexity). See Remark 9.2 for further details.

Note that Theorem 6 is derived from a similar statement for unambiguous PCIPs w.r.t. encoded provers. Indeed, in the iterative construction of Theorem 1, the batch verification we use is applied to PCIPs. We therefore focus on describing a batch verification procedure for unambiguous PCIPs w.r.t. encoded provers. That is, given such a PCIP for $\mathcal{L}$, we design an *unambiguous* PCIP *w.r.t. encoded provers* for simultaneously verifying the membership of k inputs $(x_1, \ldots, x_k)$ in $\mathcal{L}$. For our applications, it is important that the batched protocol is itself an unambiguous PCIP. We can show that this batch verification for PCIPs implies Theorem 6. See Sections 6 and 9 for formal statements and proofs.

Remark 2.8 (Controlling the query complexity)**.** *In this overview, we focus on the bounding the communication complexity of the batched* PCIP *(in particular, the communication will grow by a multiplicative factor that is only poly-logarithmic in k). We note, however, that batching*

will degrade the query complexity and verification time by a multiplicative factor of k. In the iterative construction of Theorem 1, this becomes a problem, because eventually, after repeated batchings, the query complexity will no longer be sublinear in the transcript length (we need sublinear query complexity for efficient batching, see below). This can be resolved, however, using a "query reduction" transformation (see Proposition 4.15 and Lemma 8.2). Thus, the iterative construction of Theorem 1 repeatedly uses a PCIP*-batch verification step, followed by a* PCIP*-query-reduction step (which also reduces the verifier's runtime), gradually obtaining powerful* PCIP*s (and interactive proofs) for longer and longer computations.*

Soundness for d Deviations. Let $(\mathcal{P}, \mathcal{V})$ be an unambiguous PCIP for $\mathcal{L}$. Recall that in the UP batch verification, we began by constructing a sound batch verification protocol for provers that only deviate on d of the k inputs. We later use this protocol for "deviation amplification" (see above). The ideas translate to the UIP setting, where we use $(\mathcal{P}, \mathcal{V})$ to construct a deviation amplification protocol $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$. The high level idea is as follows: The protocol starts with ℓ rounds that correspond to the ℓ rounds of the "base" protocol. In each round i, for each $j \in [k]$, let $\alpha_j^{(i)} \in \{0,1\}^a$ be the message that the (prescribed) "base" prover $\mathcal{P}$ would send on input x_j in round i given randomness $\beta^{(1)}, \ldots, \beta^{(i-1)}$ (which $\mathcal{V}_{\mathsf{amplify}}$ sent in previous rounds). The prover $\mathcal{P}_{\mathsf{amplify}}$ constructs a $k \times a$ matrix $A^{(i)}$, whose rows are the messages $(\alpha_1^{(i)}, \ldots, \alpha_k^{(i)})$, and sends its checksum $chksum^{(i)} \in \{0,1\}^{O(d \cdot \log k) \times a}$ to the verifier. The verifier receives $\widetilde{chksum}^{(i)}$, and sends random coins $\beta^{(i)}$ as sent by $\mathcal{V}$ in the base protocol (the same random coins are used for all k inputs).

After these ℓ rounds, $\mathcal{V}_{\mathsf{amplify}}$ chooses random coins for $\mathcal{V}$'s query/decision phase, sends the queries $S \subset [\ell] \times [a]$ to $\mathcal{P}_{\mathsf{amplify}}$, and receives answers $\{\phi_j : S \to \{0,1\}\}$ to those queries for each of the k base protocols. $\mathcal{V}_{\mathsf{amplify}}$ accepts if and only if: (i) $\mathcal{V}$ would accept the answers in all k protocols, and (ii) the answers are consistent with the checksums sent in rounds $1, \ldots, \ell$. Note that running these checks requires reading the values in $\{\phi_j\}$ in their entirety ($(k \cdot q)$ queries), and also making $(d \cdot q)$ queries into the transcript in rounds $1, \ldots, \ell$ to verify the checksum.

The proof of soundness against a d-deviation cheating prover is similar to the analogous proof for the UP batch verification: When a d-deviation prover sends the robust checksum value $\widetilde{chksum}^{(i)}$, it implicitly commits to messages in all k of the protocols. Thus, if $\widetilde{\mathcal{P}}$ could get $\mathcal{V}_{\mathsf{amplify}}$ to accept, we could derive a cheating prover for the base protocol, breaking its soundness. We note that it is critically important for this argument that $\widetilde{\mathcal{P}}$ sends $\widetilde{chksum}^{(i)}$ (and commits to the messages in round i), before it knows the random coins $\beta^{(i)}$ that will be chosen by the verifier for round i.

Detecting Many Deviations. As in the UP case, we leverage soundness against d deviations to amplify a cheating prover's deviations from the prescribed strategy, and obtain a sound batch verification (without any assumptions on the number of deviations). Here too, a cheating prover $\widetilde{\mathcal{P}}$ is faced with a choice. It can deviate from the prescribed strategy on d or fewer of the inputs, but then the verifier will reject w.h.p. (by soundness against d deviations). So $\widetilde{\mathcal{P}}$ may well choose to deviate on more than d of the inputs. Suppose this is the case, and there exists a subset $J^* \subseteq [k]$ of at least d of the statements, such that for every $j \in J^*$, the query answers in ϕ_j are not consistent with the prescribed strategy. The verifier $\mathcal{V}_{\mathsf{Batch}}$ would like to detect this.

Recall that in the UP batch verification, this was simple: the verifier could pick a set J_1 of $O(k/d)$ of the statements, and request the "full proof" for the statements in J_1. Here, however, it is not sufficient to ask $\widetilde{\mathcal{P}}$ to send the entire transcript for those statements. To see this, suppose that

for $j^* \in (J^* \cap J_1)$, the values in ϕ_{j^*} are not consistent with the prescribed strategy on the chosen random coins $(\beta^{(1)}, \ldots, \beta^{(\ell)})$. Unambiguity of $(\mathcal{P}, \mathcal{V})$ does not guarantee that *every* transcript that is consistent with ϕ_{j^*} makes $\mathcal{V}$ reject (given the fixed coins $(\beta^{(1)}, \ldots, \beta^{(\ell)})$). Indeed, since the coins are already fixed, there may well be many possible transcripts that make $\mathcal{V}$ accept and are consistent with ϕ_{j^*}. Thus, if all $\mathcal{V}_{\mathsf{Batch}}$ did was ask $\mathcal{P}_{\mathsf{Batch}}$ to send an accepting transcript consistent with ϕ_{j^*}, then $\widetilde{\mathcal{P}}$ could find such a transcript, and $\mathcal{V}_{\mathsf{Batch}}$ would not detect that there was a deviation in the $j^{*\text{th}}$ statement.

To make soundness go through, we design an *interactive protocol* $(\mathcal{P}_1, \mathcal{V}_1)$ for verifying that ϕ_{j^*} is consistent with the prescribed strategy on input x_{j^*} and random coins $(\beta^{(1)}, \ldots, \beta^{(\ell)})$. We obtain a sound batch verification by running this protocol for each statement x_{j^*} with $j^* \in J_1$. Loosely speaking the protocol goes as follows. First we ask $\widetilde{\mathcal{P}}$ to send the entire transcript for x_{j^*} given coins $(beta^{(1)}, \ldots, \beta^{(\ell)})$, and let $\mathcal{V}_{\mathsf{Batch}}$ verify that this transcript is consistent with ϕ_{j^*} (and makes $\mathcal{V}$ accept). Let $(\tilde{\alpha}^{(1)}, \ldots \tilde{\alpha}^{(\ell)})$ be the prover messages in this transcript. Now $\mathcal{V}_1$ simulates $\mathcal{V}$ in ℓ parallel executions of the original PCIP $(\mathcal{P}, \mathcal{V})$. At execution i, the protocol $(\mathcal{P}, \mathcal{V})$ is simulated from round i as a continuation of the i^{th} prefix of the transcript sent by $\widetilde{\mathcal{P}}$ (namely, assuming that the first $i-1$ verifier messages were $(\beta^{(1)}, \ldots, \beta^{(i-1)})$ and the first i prover messages were $(\tilde{\alpha}^{(1)}, \ldots \tilde{\alpha}^{(i)})$). It is important that the verifier uses fresh randomness $(\gamma^{(i)}, \ldots, \gamma^{(\ell)})$ for the remaining rounds (the random strings $(\gamma^{(1)}, \ldots, \gamma^{(\ell)})$ could be shared among the parallel simulations). Soundness follows, since if the transcript sent by $\widetilde{\mathcal{P}}$ first deviates from the prescribed proof at round i^* then, by the definition of unambiguity, $\mathcal{V}$ is likely to reject in the corresponding simulation of $(\mathcal{P}, \mathcal{V})$ (where $i = i^*$).

A Sound Batch Verification. Using the consistency-checking protocol $(\mathcal{P}_1, \mathcal{V}_1)$, we can obtain a sound batch verification theorem. In the batch verification protocol $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$, the prover and verifier first run the deviation amplification protocol. Then, $\mathcal{V}_{\mathsf{Batch}}$ picks at random the set J_1 of $O(k/d)$ of the instances, and the prover and verifier run the above protocol $(\mathcal{P}_1, \mathcal{V}_1)$ explicitly on each $j^* \in J_1$. Taking the number of rounds ℓ to be a constant, $(\mathcal{P}_1, \mathcal{V}_1)$ is not much more expensive than $(\mathcal{P}, \mathcal{V})$, and this yields a batch verification protocol whose communication only grows by a factor of roughly $\sqrt{k}$.

To improve the dependence on k, and obtain the result claimed in Theorem 6, we recurse as in the UP batch verification theorem. $\mathcal{P}_{\mathsf{Batch}}$ and $\mathcal{V}_{\mathsf{Batch}}$ use the protocol $(\mathcal{P}_1, \mathcal{V}_1)$ as a "base protocol", and run the deviation amplification protocol to amplify the number of deviations *within* J_1 to at least d instances (note that $\mathcal{P}_{\mathsf{Batch}}$ and $\mathcal{V}_{\mathsf{Batch}}$ never explicitly run the protocol $(\mathcal{P}_1, \mathcal{V}_1)$). Now $\mathcal{V}_{\mathsf{Batch}}$ picks a smaller set $J_2 \subset J_1$ of size $O(|J_1|/d)$, and the prover and verifier recurse again and again until they obtain a set J_{final} of size $O(d)$ and a protocol $(\mathcal{P}_{final}, \mathcal{V}_{final})$ that will w.h.p. reject at least one of the instances in J_{final}. At the "base" of this recursion, the prover and verifier explicitly run the protocol $(\mathcal{P}, \mathcal{V})$ on every instance in the set J_{final}.

While the complexity of the "base protocol" grows by a factor of ℓ in every level of the recursion, the set of instances under consideration shrinks by a factor of d. Taking $d = k^\tau$ for a constant $0 < \tau \ll 1$, we only have a constant number of levels in the recursion, and the final consistency-checking protocol $(\mathcal{P}_{final}, \mathcal{V}_{final})$ is only roughly $\ell^{O(1/\tau)}$ times more expensive than the base protocol (throughout we think of ℓ as a constant). The resulting batch verification protocol $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$ has roughly $O(\ell)$ rounds, communication complexity $(\mathsf{poly}(\ell) \cdot k^\tau \cdot c)$, prover runtime $(\mathsf{poly}(\ell) \cdot k \cdot \mathcal{P}\mathsf{time})$ and verifier runtime $(\mathsf{poly}(\ell) \cdot k \cdot \mathcal{V}\mathsf{time})$. The query complexity is $(\mathsf{poly}(\ell) \cdot k \cdot q)$.

The soundness error grows linearly with the number of levels in the recursion, yielding $O(\epsilon)$

soundness. For simplicity, we assume here that ϵ is larger than $(1/k^{2\tau})$ so we can take each set J_m in $J_1, \ldots, J_{final}$ to be of size $O(\log(1/\epsilon) \cdot |J_{m-1}|/d)$, and still have a constant number of levels in the recursion. The size bound on J_m guarantees that the probability that any set in the sequence $J_1, \ldots, J_{final}$ "misses" the deviating instances is smaller than ϵ.

Remark 2.9 (Maintaining unambiguity)**.** *We note that in the statement of Theorem 6, and to use the batch verification protocol repeatedly in the iterative construction of Theorem 1, we need to show that the* PCIP $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$ *is itself* unambiguous *w.r.t. encoded provers (above we sketched why the protocol is sound). This presents some additional challenges. See the full proof in Section 6 for details.*

3 Preliminaries

For a string $x \in \Sigma^n$ and an index $i \in [n]$, we denote by $x|_i \in \Sigma$ the i^{th} entry in x. If $I \subseteq [n]$ is a set then we denote by $x|_I$ the sequence of entries in x corresponding to coordinates in I. For a matrix $A \in \Sigma^{n \times m}$ and an index $i \in [m]$, we denote by $A|_i \in \Sigma^m$ the i^{th} column of A. For $I \subseteq [m]$, we denote by $A|_I$ the restriction of A to the columns in I.

Let $x, y \in \Sigma^n$ be two strings of length $n \in \mathbb{N}$ over a (finite) alphabet Σ. We define the (absolute) distance of x and y as $\Delta(x, y) \stackrel{\text{def}}{=} |\{x_i \neq y_i : i \in [n]\}|$. If $\Delta(x, y) \leq \varepsilon \cdot n$, then we say that x is ε-close to y, and otherwise we say that x is ε-far from y. We define the distance of x from a (non-empty) set $S \subseteq \Sigma^n$ as $\Delta(x, S) \stackrel{\text{def}}{=} \min_{y \in S} \Delta(x, y)$. If $\Delta(x, S) \leq \varepsilon \cdot n$, then we say that x is ε-close to S and otherwise we say that x is ε-far from S. We extend these definitions from strings to functions by identifying a function with its truth table.

3.1 Multivariate Polynomials and Low Degree Testing

In this section we recall some important facts on multivariate polynomials (see [Sud95] for a far more detailed introduction). A basic fact, captured by the Schwartz-Zippel lemma is that low degree polynomials cannot have too many roots.

Lemma 3.1 (Schwartz-Zippel Lemma)**.** *Let $P : \mathbb{F}^m \to \mathbb{F}$ be a* non-zero *polynomial of total degree d. Then,*

$$\Pr_{r \in \mathbb{F}^m}\left[P(r) = 0\right] \leq \frac{d}{|\mathbb{F}|}.$$

An immediate corollary of the Schwartz-Zippel Lemma is that two distinct polynomials $P, Q : \mathbb{F}^m \to \mathbb{F}$ of total degree d may agree on at most a $\frac{d}{|\mathbb{F}|}$-fraction of their domain $\mathbb{F}^m$.

Throughout this work we consider fields in which operations can be implemented efficiently (i.e., in poly-logarithmic time in the field size). Formally we define such fields as follows.

Definition 3.2. *We say that an ensemble of finite fields $\mathbb{F} = (\mathbb{F}_n)_{n \in \mathbb{N}}$ is* constructible *if elements in $\mathbb{F}_n$ can be represented by $O(\log(|\mathbb{F}_n|))$ bits and field operations (i.e., addition, subtraction, multiplication, inversion and sampling random elements) can all be performed in* $\mathsf{polylog}(|\mathbb{F}_n|)$ *time given this representation.*

A well known fact is that for every $S = S(n)$, there exists a *constructible* field ensemble of size $O(S)$ and its representation can be found in $\mathsf{polylog}(S)$ time (see, e.g., [Gol08, Appendix G.3] for details).

In the following we consider polynomials over a constructible field. In this work we will use the following well-known local testability and decodability properties of polynomials.

Lemma 3.3 (Self-Correction Procedure (cf. [GS92, Sud95])**.** *Let $\mathbb{F}$ be a constructible field ensemble. Let $\delta < 1/3$, $\varepsilon \in (0,1]$, and $d, m \geq 1$. There exists an algorithm that, given input $x \in \mathbb{F}^m$ and oracle access to an m-variate function $P : \mathbb{F}^m \to \mathbb{F}$ that is δ-close to a polynomial P' of total degree d, runs in time $O(d \cdot m \cdot \log(|\mathbb{F}|) \cdot \log(1/\varepsilon))$ makes $O(d \cdot \log(1/\varepsilon))$ oracle queries and outputs $P'(x)$ with probability $1 - \varepsilon$. Furthermore, if P itself has total degree d, then given $x \in \mathbb{F}^m$, the algorithm outputs $P(x)$ with probability 1.*

Lemma 3.4 (Low Degree Test ([RS96, Sud95, AS03]) see also [Gol16])**.** *Let $\mathbb{F}$ be a constructible field ensemble. Let $\delta \in (0, 1/2)$, $\varepsilon \in (0,1]$ and $d, m \in \mathbb{N}$, such that $d \leq |\mathbb{F}|/2$. There exists a randomized algorithm that, given oracle access to an m-variate function $P : \mathbb{F}^m \to \mathbb{F}$, runs in time $(d \cdot m \cdot \log(|\mathbb{F}|) \cdot \mathsf{poly}(1/\delta) \cdot \log(1/\varepsilon))$, makes $(d \cdot \mathsf{poly}(1/\delta) \cdot \log(1/\varepsilon))$ oracle queries and:*

1. *Accepts every function that is a polynomial of* total *degree d with probability 1; and*
2. *Rejects functions that are δ-far from every polynomial of total degree d with probability at least $1 - \varepsilon$.*

We will also need a variant of the low degree test that tests the individual degree of the polynomial (rather than total degree). Such a test is implicit in [GS06, Section 5.4.2] (see also [GR15, Theorem A.9]).

Lemma 3.5 (Individual Degree Test)**.** *Let $\mathbb{F}$ be a constructible field ensemble. Let $\delta \in (0, 1/2)$, $\varepsilon \in (0,1]$ and $d, m \in \mathbb{N}$ such that $d \cdot m < |\mathbb{F}|/10$. There exists a randomized algorithm that, given oracle access to an m-variate polynomial $P : \mathbb{F}^m \to \mathbb{F}$, runs in time $(d \cdot m \cdot \log(|\mathbb{F}|) \cdot \mathsf{poly}(1/\delta) \cdot \log(1/\varepsilon))$, makes $(d \cdot m \cdot \mathsf{poly}(1/\delta) \cdot \log(1/\varepsilon))$ oracle queries and:*

1. *Accepts every function that is a polynomial of* individual *degree d with probability 1; and*
2. *Rejects functions that are δ-far from every polynomial of individual degree d with probability at least $1 - \varepsilon$.*

3.2 Low Degree Extension

Let $\mathbb{H}$ be a finite field and $\mathbb{F} \supseteq \mathbb{H}$ be an extension field of $\mathbb{H}$. Fix an integer $m \in \mathbb{N}$. A basic fact is that for every function $\phi : \mathbb{H}^m \to \mathbb{F}$, there exists a unique extension of ϕ into a function $\hat{\phi} : \mathbb{F}^m \to \mathbb{F}$ (which agrees with ϕ on $\mathbb{H}^m$; i.e., $\hat{\phi}|_{\mathbb{H}^m} \equiv \phi$), such that $\hat{\phi}$ is an m-variate polynomial of individual degree at most $|\mathbb{H}| - 1$. Moreover, there exists a collection of $|\mathbb{H}|^m$ functions $\{\hat{\tau}_x\}_{x \in \mathbb{H}^m}$ such that each $\hat{\tau}_x : \mathbb{F}^m \to \mathbb{F}$ is the m-variate polynomial of degree $|\mathbb{H}| - 1$ in each variable defined as:

$$\hat{\tau}_x(z) \stackrel{\text{def}}{=} \prod_{i \in [m]} \prod_{h \in \mathbb{H} \setminus \{x_i\}} \frac{z_i - h}{x_i - h}.$$

and for every function $\phi : \mathbb{H}^m \to \mathbb{F}$ it holds that

$$\hat{\phi}(z_1, \ldots, z_m) = \sum_{x \in \mathbb{H}^m} \hat{\tau}_x(z_1, \ldots, z_m) \cdot \phi(x).$$

The function $\hat{\phi}$ is called the *low degree extension* of ϕ (with respect to $\mathbb{F}$, $\mathbb{H}$ and m).

We also define the individual degree $|\mathbb{H}|-1$ polynomial $\hat{\tau} : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ as:

$$\hat{\tau}(x,z) \stackrel{\text{def}}{=} \prod_{i\in[m]} \prod_{h\in\mathbb{H}\setminus\{0\}} \frac{z_i - x_i - h}{h}. \tag{1}$$

Observe that for every $x \in \mathbb{H}^m$ it holds that $\hat{\tau}(x,\cdot) \equiv \hat{\tau}_x(\cdot)$.

From Eq. (1) we can immediately deduce the following proposition.

Proposition 3.6 (Cf., e.g., [Rot09, Proposition 3.2.1])**.** *Let $\mathbb{H} = (\mathbb{H}_n)_n$ and $\mathbb{F} = (\mathbb{F}_n)_n$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$.*

There exists a Turing machine that on input n outputs the polynomial $\hat{\tau} : \mathbb{F}^m \times \mathbb{F}^m \to \mathbb{F}$ defined above, represented as an arithmetic circuit over $\mathbb{F}$, in time $\mathsf{poly}(|\mathbb{H}|, m, \log|\mathbb{F}|)$ and space $O(\log(|\mathbb{F}|)+\log(m))$.

Moreover, the arithmetic circuit $\hat{\tau}$ can be evaluated in time $\mathsf{poly}(|\mathbb{H}|, m, \log(|\mathbb{F}|))$ and space $O(\log(|\mathbb{F}|)+\log(m))$. Namely, there exists a Turing machine with the above time and space bounds that given an input pair $(x,z) \in \mathbb{F}^m \times \mathbb{F}^m$ outputs $\hat{\tau}(x,z)$.

Proposition 3.7. *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$.*

Let $\phi : \mathbb{H}^m \to \mathbb{F}$ and suppose that ϕ can be evaluated by a Turing Machine in time t and space s. Then, there exists a Turing machine that, given as an input a point $z \in \mathbb{F}^m$, runs in time $|\mathbb{H}|^m \cdot (\mathsf{poly}(|\mathbb{H}|, m, \log(|\mathbb{F}|)) + O(t))$ and space $O(m \cdot \log(|\mathbb{H}|) + s + \log(|\mathbb{F}|))$ and outputs the value $\hat{\phi}(z)$ where $\hat{\phi}$ is the unique low degree extension of ϕ (with respect to $\mathbb{H}, \mathbb{F}, m$).

Proof. The Turing machine computes

$$\hat{\phi}(z) = \sum_{x\in\mathbb{H}^m} \hat{\tau}_x(z) \cdot \phi(x)$$

by generating and evaluating $\hat{\tau}_x(z)$ as in Proposition 3.6. □

The Low Degree Extension as an Error Correcting Code. The low degree extension can be viewed as an error correcting code applied to bit strings. Formally, let $\mathbb{F}$ be an extension field of a base field $\mathbb{H}$. Fix some canonical ordering of the elements in $\mathbb{H}$. For every integer $n \in \mathbb{N}$ we identify the set $[n]$ with the set $\mathbb{H}^{\log_{|\mathbb{H}|}(n)}$ by taking the representation of $i \in [n]$ in base $|\mathbb{H}|$.

Consider the function $\mathsf{LDE}_{\mathbb{F},\mathbb{H}} : \{0,1\}^n \to \mathbb{F}^{|\mathbb{F}|^m}$, where $m = \log_{|\mathbb{H}|}(n)$, that given a string $\phi \in \{0,1\}^n$, views ϕ as a function $\phi : \mathbb{H}^m \to \{0,1\}$, by identifying $[n]$ with $\mathbb{H}^m$ as above, and outputs the truth table of the low degree extension $\hat{\phi} : \mathbb{F}^m \to \mathbb{F}$ of ϕ, represented as an $|\mathbb{F}|^m$ dimensional vector. The Schwartz-Zippel Lemma (Lemma 3.1) shows that this code has relative distance $1 - \frac{m\cdot(|\mathbb{H}|-1)}{|\mathbb{F}|}$.

As usual in the context of error correcting codes, we will sometimes use $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ to denote the set of codewords $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m} \stackrel{\text{def}}{=} \left\{\hat{\phi} \text{ s.t. } \phi : \mathbb{H}^m \to \{0,1\}\right\}$. That is, $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ contains m-variate polynomials of individual degree $(|\mathbb{H}|-1)$, whose values on the inputs in $\mathbb{H}^m$ are all in $\{0,1\}$.

A Low-Degree Extension for Multiple Strings. We will also need to extend the $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ encoding to encode multiple strings (of varying lengths). A natural way of doing so is by simply concatenating the strings and applying $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ to the concatenated string. However, we take a slightly different approach, which will be useful both for composing encodings of individual strings to a single joint encoding, and for decomposing such a joint encoding into individual encodings. We remark that this approach leverages the fact that the low degree extension is a tensor code [Wol65].

Given strings $x_1 \in \{0,1\}^{n_1}, \ldots, x_k \in \{0,1\}^{n_k}$, we define an encoding $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$ as follows. Let $m_k = \log_{|\mathbb{H}|}(k)$, $m^{(i)} = \log_{|\mathbb{H}|}(n_i)$, $m_{\max} = \max_{i \in [k]} m^{(i)}$ and $m = m_k + m_{max}$. We identify $[k]$ with $\mathbb{H}^{m_k}$ and identify $[\max(|x_i|)]$ with $\mathbb{H}^{m_{max}}$ by viewing the respective integers in base $|\mathbb{H}|$. Let $P : \mathbb{H}^m \to \{0,1\}$ be a function such that for every $z \in \mathbb{H}^{m_k}$, and every $(w, y) \in \mathbb{H}^{m^{(z)}} \times \mathbb{H}^{m - m_k - m^{(z)}}$, it holds that: $P(z, w, y)$ is equal to the w^{th} bit of x_z if $y = 0^{m - m_k - m^{(z)}}$, and $P(z, w, y) = 0$ otherwise (i.e., if $y \neq 0^{m - m_k - m^{(z)}}$). We define $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$ as the low degree extension of P with respect to $\mathbb{F}$, $\mathbb{H}$ and m.

Proposition 3.8. *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$ and $|\mathbb{H}| \geq \log(|\mathbb{F}|)$. There exist procedures* Compose *and* Decompose *as follows:*

- **Compose:** *Given oracle access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1), \ldots, \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_k)$ and a point $z \in \mathbb{F}^m$, where m is as above, the procedure* Compose *makes a single query to each $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_i)$ (k queries in total), runs in time $k \cdot \mathsf{poly}(|\mathbb{H}|, \sum_i \log_{|\mathbb{H}|}(n_i), \log(k))$ and outputs the z^{th} entry of $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$.*

- **Decompose:** *Given oracle access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$, some $i \in [k]$ and a point $w \in \mathbb{F}^{m^{(i)}}$, where $m^{(i)}$ is as above, the procedure* Decompose *makes a single oracle query, runs in time $\mathsf{poly}(|\mathbb{H}|, \log(k), \sum_i \log_{|\mathbb{H}|}(n_i))$ and outputs the w^{th} entry of $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_i)$.*

Proof. Let $x_1 \in \{0,1\}^{n_1}, \ldots, x_k \in \{0,1\}^{n_k}$. Let $m_k = \log_{|\mathbb{H}|}(k)$, $m^{(i)} = \log_{|\mathbb{H}|}(n_i)$ for every $i \in [k]$, $m_{\max} = \max_i m^{(i)}$ and $m = m_k + m_{\max}$.

For every $z \in \mathbb{H}^{m_k}$, let $\hat{P}_z : \mathbb{F}^{m^{(i)}} \to \mathbb{F}$ be the polynomial $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_z)$, of individual degree $(|\mathbb{H}| - 1)$, where we identify $\mathbb{H}^{m_k}$ with $[k]$ as above. Let $\hat{P} : F^m \to \mathbb{F}$ be the polynomial $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$, of individual degree $(|\mathbb{H}| - 1)$.

Claim 3.8.1. *For every $z \in \mathbb{H}^{m_k}$, the polynomials $\hat{P}_z(\cdot)$ and $\hat{P}(z, \cdot, 0^{m - m_k - m^{(z)}})$ are identical (over their domain $\mathbb{F}^{m^{(z)}}$).*

Proof. For every $z \in \mathbb{H}^{m_k}$ and every $w \in \mathbb{F}^{m^{(z)}}$ it holds that

$$\begin{aligned}
\hat{P}(z, w, 0^{m-m_k-m^{(z)}}) &= \sum_{(z',w',y') \in \mathbb{H}^{m_k} \times \mathbb{H}^{m^{(z)}} \times \mathbb{H}^{m-m_k-m^{(z)}}} \tau_{(z',w',y')}(z, w, 0^{m-m_k-m^{(z)}}) \cdot \hat{P}(z', w', y') \\
&= \sum_{(z',w',y') \in \mathbb{H}^{m_k} \times \mathbb{H}^{m^{(z)}} \times \mathbb{H}^{m-m_k-m^{(z)}}} \tau_{z'}(z) \cdot \tau_{w'}(w) \cdot \tau_{y'}(0^{m-m_k-m^{(z)}}) \cdot \hat{P}(z', w', y') \\
&= \sum_{w' \in \mathbb{H}^{m^{(z)}}} \tau_{w'}(w) \cdot \hat{P}(z, w', 0^{m-m_k-m^{(z)}}) \\
&= \sum_{w' \in \mathbb{H}^{m^{(z)}}} \tau_{w'}(w) \cdot x_z[w'] \\
&= \sum_{w' \in \mathbb{H}^{m^{(z)}}} \tau_{w'}(w) \cdot \hat{P}_z(w') \\
&= \hat{P}_z(w),
\end{aligned}$$

where $x_z[w']$ refers to the w'^{th} entry of x_z (where we view z as an integer in $[k]$ and w' as an integer in $[n_i]$). Hence, $\hat{P}(z, \cdot, 0^{m-m_k-m^{(z)}})$ and $\hat{P}_z(\cdot)$, which are individual degree $|\mathbb{H}| - 1$ polynomials, agree on $\mathbb{H}^{m^{(z)}}$ and so, by the Schwartz-Zippel Lemma (Lemma 3.1), they must be equal. □

Hence, for every $i \in [k]$ and $w \in \mathbb{F}^{m^{(i)}}$, the value $\hat{P}_i(w)$ can be retrieved by using a single query to $\hat{P}$ and this query can be generated in time $\mathsf{poly}(|\mathbb{H}|, \log(k), \sum_i \log_{|\mathbb{H}|}(n_i))$.

For the $\mathsf{Compose}$ procedure, observe that for every $(z, w) \in \mathbb{F}^{m_k} \times F^{m-m_k}$ it holds that

$$\begin{aligned}
\hat{P}(z, w) &= \sum_{(z',w') \in \mathbb{H}^{m_k} \times \mathbb{H}^{m-m_k}} \tau_{(z',w')}(z, w) \cdot \hat{P}(z', w') \\
&= \sum_{z' \in \mathbb{H}^{m_k}} \tau_{z'}(z) \sum_{w' \in \mathbb{H}^{m-m_k}} \tau_{w'}(w) \cdot \hat{P}(z', w') \\
&= \sum_{z' \in \mathbb{H}^{m_k}} \tau_{z'}(z) \sum_{w' \in \mathbb{H}^{m^{(z)}}} \tau_{(w', 0^{m-m_k-m^{(z)}})}(w) \cdot \hat{P}(z', w', 0^{m-m_k-m^{(z)}}) \\
&= \sum_{z' \in \mathbb{H}^{m_k}} \tau_{z'}(z) \cdot \tau_{0^{m-m_k-m^{(z)}}}(w|_{[m^{(z)}+1, m-m_k]}) \cdot \sum_{w' \in \mathbb{H}^{m^{(z)}}} \tau_{w'}(w|_{[1, m^{(z)}]}) \cdot \hat{P}_{z'}(w') \\
&= \sum_{z' \in \mathbb{H}^{m_k}} \tau_{z'}(z) \cdot \tau_{0^{m-m_k-m^{(z)}}}(w|_{[m^{(z)}+1, m-m_k]}) \cdot \hat{P}_{z'}(w|_{[1, m^{(z)}]}),
\end{aligned}$$

where $w|_{[a,b]}$ denotes the projection of w to the interval $[a, b]$. Thus, the value $\hat{P}(z, w)$ can be retrieved by making a single query to each $\hat{P}_{z'}$. The overall time for running this procedure is $k \cdot \mathsf{poly}(|\mathbb{H}|, \sum m^{(i)}, \log(k))$.

□

The following proposition shows an efficient procedure to check that the suffix of a string encoded under the $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ encoding is all-zeros.

Proposition 3.9. *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$. There exists a randomized algorithm $\mathcal{A}$ that is given oracle access to a string $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$, where*

$x \in \{0,1\}^n$, and a parameter $k \in [n]$, runs in time $O(|\mathbb{H}| \cdot \log_{|\mathbb{H}|}(n))$ and makes $O(|\mathbb{H}| \cdot \log_{|\mathbb{H}|}(n))$ queries such that if the k-bit suffix of x is zeros then $\mathcal{A}$ accepts and otherwise $\mathcal{A}$ rejects with probability $1 - \frac{|\mathbb{H}| \cdot \log_{|\mathbb{H}|}(n)}{|\mathbb{F}|}$.

Proof. Let $m = \log_{|\mathbb{H}|}(n)$ and let $k \in [n]$ be associated with the string $(h_1, \ldots, h_m) \in \mathbb{H}^m$ where we identify $[n]$ with $\mathbb{H}^m$ as described above. For every $h > h_1$ (here we use the order on elements in $\mathbb{H}$ that was fixed above) the algorithm $\mathcal{A}$ checks that the polynomial $\big(\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)\big)(h, \cdot, \ldots, \cdot)$ is the all-zeros polynomial by querying it at a random location (in $\mathbb{F}^{m-1}$) and checking that the result is 0. The algorithm also recurses on input $\big(\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)\big)(h_1, \cdot, \ldots, \cdot)$ w.r.t the parameter $k' = (h_2, \ldots, h_m)$.

If $x[k, \ldots, n] = 0^{n-k}$ then the algorithm always accepts. Otherwise, $\exists i \in [k+1, \ldots, n]$ such that $x_i \neq 0$. Let $(h'_1, \ldots, h'_m) \in \mathbb{H}^m$ be associated with i. Let j be the minimal index on which $(h_1, \ldots, h_m)$ differs from $(h'_1, \ldots, h'_m)$ (such an index exists since $i > k$). After j recursive invocations, the algorithm will consider the polynomial $\big(\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)\big)(h_1, \ldots, h_{j-1}, \cdot, \ldots, \cdot)$ which is a non-zero polynomial with individual degree $|\mathbb{H}| - 1$. Hence, by the Schwartz-Zippel Lemma (Lemma 3.1) it will reject with probability $1 - \frac{m \cdot (|\mathbb{H}|-1)}{|\mathbb{F}|}$. □

The following proposition allows us to check if an LDE codeword is zero except for a small set of coordinates.

Proposition 3.10. *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$, where $|\mathbb{H}| \geq \log(|\mathbb{F}|)$ and let $m = m(n)$. There exists a randomized algorithm $\mathcal{A}$ that is given as explicit input a set $S \subseteq \mathbb{H}^m$ and as implicit input an individual degree $|\mathbb{H}| - 1$ polynomial $P : \mathbb{F}^m \to \mathbb{F}$. The algorithm runs in time $|S| \cdot \mathsf{poly}(|\mathbb{H}|, m, \log(|\mathbb{F}|))$ and makes at most $|S| + 1$ queries. If $P|_{\mathbb{H}^m \setminus S} \equiv 0$, then $\mathcal{A}$ accepts and otherwise, with probability $1 - \frac{|\mathbb{H}| \cdot m}{|\mathbb{F}|}$ it rejects.*

Furthermore, the queries that $\mathcal{A}$ makes depend only on the set S and its random coins.

Proof. Define $Q(z) \stackrel{\text{def}}{=} \sum_{s \in S} \hat{\tau}_s(z) \cdot P(s)$, where $\hat{\tau}$ is as defined in Section 3.2. The verifier chooses a random point $r \in \mathbb{F}^m$ and checks that $Q(r) = P(r)$, where P is the input polynomial and in order to compute Q the verifier reads $P(s)$ for all $s \in S$. If the check passes then the verifier accepts and otherwise it rejects.

For completeness, observe that if $P|_{\mathbb{H}^m \setminus S} \equiv 0$ then, using the definition of the low degree extension,

$$P(r) = \sum_{x \in \mathbb{H}^m} \hat{\tau}_x(r) \cdot P(x) = \sum_{s \in S} \hat{\tau}_s(r) \cdot P(s) = Q(r)$$

and so the verifier accepts.

For soundness, let $x^* \in \mathbb{H}^m \setminus S$ such that $P(x^*) \neq 0$. Then, since $x^* \in \mathbb{H}^m$, for every $s \in S$ it holds that $\hat{\tau}_s(x^*) = \tau_s(x^*) = 0$, where the equality to 0 follows from the fact that $x^* \notin S$. Thus,

$$Q(x^*) = \sum_{s \in S} \hat{\tau}_s(x^*) \cdot P(s) = 0$$

and so $P(x^*) \neq Q(x^*)$. Note that P and Q are *different* polynomials of total degree $(|\mathbb{H}| - 1) \cdot m$ and so, by the Schwartz-Zippel Lemma (Lemma 3.1) the verifier rejects with probability at least $1 - \frac{|\mathbb{H}| \cdot m}{|\mathbb{F}|}$, when checking whether $P(r) = Q(r)$.

As for the complexity of the algorithm, using Proposition 3.6, we can compute $Q(r)$ in time $s \cdot \mathsf{poly}(|\mathbb{H}|, m, \log(|\mathbb{F}|))$ and using only s queries to P (where the query $P(r)$ is an additional query). □

3.3 Uniformity and Constructibility

In this section we specify the notions of uniform circuits and 3CNF formulas, which will be used in our protocols. We then review a proof that polynomial-time computation can be verified by a constructible 3CNF ensemble.

Definition 3.11 (T-Uniform arithmetic circuit ensemble)**.** *Let $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ be an ensemble of arithmetic circuits over a constructible field ensemble $\mathbb{F}$ (see Definition 3.2), where C_n is over n field elements. We say that $\{C_n\}_{n\in\mathbb{N}}$ is T-*uniform *for a function $T : \mathbb{N} \to \mathbb{N}$, if there exists a Turing Machine that on input 1^n runs in time $T(n)$ and outputs C_n (observe that it also follows that C_n can be evaluated in time $(T \cdot \mathsf{polylog}(T))$).*

The following definitions consider ensembles $\{\phi_n\}_{n\in\mathbb{N}}$ of 3CNF formulas. An ensemble of 3CNFs is T-uniform if the n-th formula in the ensemble can be generated in time $T(n)$ (i.e. if the ensemble is T-uniform, as per Definition 3.11). We say that an ensemble of 3CNFs has size $s = s(n)$ if, for every $n \in \mathbb{N}$, it holds that ϕ_n is a 3CNF formula with $s(n)$ clauses on n variables. (Note that every ensemble of 3CNF formulas has size at most $O(n^3)$.)

Definition 3.12 (Constructible 3CNF Ensembles)**.** *Let $\{\mathbb{H}_n\}_{n\in\mathbb{N}}$ be a constructible ensemble of finite fields. Take $m = m(n) = \frac{\log(n)}{\log(|\mathbb{H}|)}$ (i.e., $|\mathbb{H}^m| = n$). We identify $\mathbb{H}^m$ with $[n]$ via a canonical mapping.*

We say that an ensemble of 3CNF *formulas $\psi = (\psi_n)_{n\in\mathbb{N}}$ (where $\psi_n : \{0,1\}^n \to \{0,1\}$ is on n variables) is* $\mathbb{H}$-constructible *if ψ is polynomial-time uniform, and there exists an ensemble of arithmetic circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ over $\mathbb{H}$, where $C_n : \mathbb{H}^{3m+3} \to \{0,1\}$, such that on input $i_1, i_2, i_3 \in \mathbb{H}^m$ and $b_1, b_2, b_3 \in \{0,1\}$ the circuit outputs 1 if and only if the clause:*

$$\big(x_{i_1} = b_1\big) \vee \big(x_{i_2} = b_2\big) \vee \big(x_{i_3} = b_3\big)$$

appears in ψ_n. For inputs outside of $\mathbb{H}^{3m} \times \{0,1\}^3$ the circuit C_n is identically 0.

We say that $\mathcal{C}$ specifies *the formula ψ. We require that $\mathcal{C}$ is $\mathsf{poly}(|\mathbb{H}|, m)$-uniform, can be evaluated in time $\mathsf{poly}(|\mathbb{H}|, m)$, and has individual degree $\mathsf{poly}(|\mathbb{H}|, m)$.*

We sometimes consider evaluating $\mathcal{C}$ over a constructible ensemble of extension fields $\mathbb{F} = \{\mathbb{F}_n\}$, where for every n, $\mathbb{F}_n$ is an extension field of $\mathbb{H}_n$ of size $\mathsf{poly}(|\mathbb{H}|)$. When this is the case, we take $\hat{C} : \mathbb{F}^{3m+3} \to \mathbb{F}$ to be the circuit that simply evaluates $\mathcal{C}$ over the extension field $\mathbb{F}$, and we note that $\hat{\mathcal{C}}$ also is $\mathsf{poly}(|\mathbb{H}|, m)$-uniform, can be evaluated in time $\mathsf{poly}(|\mathbb{H}|, m)$, and has individual degree $\mathsf{poly}(|\mathbb{H}|, m)$. We also refer to $\hat{\mathcal{C}}$ as specifying the formula ψ.

Proposition 3.13 (From Turing Machines to Constructible 3CNFs)**.** *Let $\mathbb{H} = (\mathbb{H}_n)_{n\in\mathbb{N}}$ be a constructible ensemble of finite fields and let $\mathcal{L}$ be a language computed by a Turing Machine running in time $T(n)$ and space $S(n)$. Then, there exists a* linear-sized $\mathbb{H}$*-constructible* 3CNF *ensemble $\psi = (\psi_n)_{n\in\mathcal{M}}$ such that:*

- *For every $x \in \mathcal{L}$, there exists a* unique *witness $w \in \{0,1\}^{O(T\cdot S)}$ such that $\psi_{|x|+|w|}(x, w) = 1$.*
- *For every $x \notin \mathcal{L}$, and for every $w \in \{0,1\}^{O(T\cdot S)}$ it holds that $\psi_{|x|+|w|}(x, w) = 0$.*

Given $x \in \mathcal{L}$, the witness w can be computed in $O(T)$ time and $O(S)$ space.

The proof of Proposition 3.13 follows from the Cook-Levin Theorem.

4 Unambiguous and Probabilistically Checkable Interactive Proofs

An interactive protocol consists of a pair $(\mathcal{P}, \mathcal{V})$ of interactive Turing machines that are run on a common input x, whose length we denote by $n = |x|$. The first machine, which is deterministic, is called *the prover* and is denoted by $\mathcal{P}$, and the second machine, which is probabilistic, is called *the verifier* and is denoted by $\mathcal{V}$.

An $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}, \Sigma)$-interactive protocol, is an interactive protocol in which, for inputs of length n, the parties interact for $\ell = \ell(n)$ rounds and each message sent from $\mathcal{P}$ to $\mathcal{V}$ (resp., $\mathcal{V}$ to $\mathcal{P}$) is in Σ^a (resp., Σ^b), where $\Sigma = \Sigma(n)$ is an alphabet (whose size may depend on n). The verifier runs in time $\mathcal{V}\mathsf{time}$ and the prover runs in time $\mathcal{P}\mathsf{time}$. We typically omit Σ from the notation and refer to $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ interactive protocols when Σ is clear from the context.

In an $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}, \Sigma)$-interactive protocol, in each round $i \in [\ell]$, first $\mathcal{P}$ sends a message $\alpha^{(i)} \in \Sigma^a$ to $\mathcal{V}$ and then $\mathcal{V}$ sends a message $\beta^{(i)} \in \Sigma^b$ to $\mathcal{P}$. At the end of the interaction $\mathcal{V}$ runs a (deterministic) Turing machine on input $\left(x, r, \left(\alpha^{(1)}, \ldots, \alpha^{(\ell)}\right)\right)$, where r is its random string and outputs the result. Abusing notation, we denote the result of the computation by $\mathcal{V}\left(x, r, \left(\alpha^{(1)}, \ldots, \alpha^{(\ell)}\right)\right)$. We also denote by $\mathcal{P}\left(x, i, \left(\beta^{(1)}, \ldots, \beta^{(i-1)}\right)\right)$ the message sent by $\mathcal{P}$ in round i given input x and receiving the messages $\beta^{(1)}, \ldots, \beta^{(i-1)}$ from $\mathcal{V}$ in rounds $1, \ldots, i-1$, respectively. We emphasize that $\mathcal{P}$'s messages depend only on the input x and on the messages that it received from $\mathcal{V}$ in previous rounds.

The communication complexity of an $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}, \Sigma)$-interactive protocol is the total number of bits transmitted. Namely, $(\ell \cdot (b + a) \cdot \log_2(|\Sigma|))$.

Public-coin Protocols. In this work we focus on public-coin interactive protocols, which are interactive protocols in which each message $\beta^{(i)}$ sent from the verifier to the prover is a uniformly distributed random string in Σ^b. At the end of the protocol, $\mathcal{V}$ decides whether to accept or reject as a function of x and the messages $\alpha^{(1)}, \beta^{(1)}, \ldots, \alpha^{(\ell)}, \beta^{(\ell)}$.

4.1 Interactive Proofs (IPs)

The classical notion of an interactive proof for a language $\mathcal{L}$ is due to Goldwasser, Micali and Rackoff [GMR89].

Definition 4.1 (ε-sound $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IP [GMR89])**.** *An $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-interactive protocol $(\mathcal{P}, \mathcal{V})$ (as above) is an ε-sound $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-Interactive Proof (IP) for $\mathcal{L}$ if:*

- **Completeness:** *For every $x \in \mathcal{L}$, if $\mathcal{V}$ interacts with $\mathcal{P}$ on common input x, then $\mathcal{V}$ accepts with probability 1.*[11]
- **ε-Soundness:** *For every $x \notin \mathcal{L}$ and every (computationally unbounded) cheating prover strategy $\widetilde{\mathcal{P}}$, the verifier $\mathcal{V}$ accepts when interacting with $\tilde{P}$ with probability less than $\varepsilon(|x|)$, where $\varepsilon = \varepsilon(n)$ is called the soundness error of the proof-system.*

We remark that our definition of interactive proofs emphasizes the parameters of the proof-system (to a higher degree than is commonly done in the literature). This is mainly because throughout this work we apply transformations to interactive proofs and we need to carefully keep track of the effect of these transformations on each one of these parameters.

[11] One could allow an error also in the completeness condition. For simplicity, and since all our protocols do not have such an error, we require perfect completeness.

4.2 Unambiguous IPs

In this work we introduce a variant of interactive proofs, which we call *unambiguous interactive proofs*, in which the *honest* prover strategy is defined for *every* x (i.e., also for $x \notin \mathcal{L}$) and the verifier is required to reject when interacting with any cheating prover that deviates from the prescribed honest prover strategy at any point of the interaction.

More formally, if $(\mathcal{P}, \mathcal{V})$ is an interactive protocol, and $\widetilde{\mathcal{P}}$ is some arbitrary (cheating) strategy, we say that $\widetilde{\mathcal{P}}$ deviates from the protocol at round i^* if the message sent by $\widetilde{\mathcal{P}}$ in round i^* differs from the message that $\mathcal{P}$ would have sent given the transcript of the protocol thus far. In other words, if the verifier sent the messages $\beta^{(1)}, \ldots, \beta^{(i-1)}$ in rounds $1, \ldots, (i-1)$ respectively, we say that $\widetilde{\mathcal{P}}$ deviates from the protocol at round i^* if $\widetilde{\mathcal{P}}\left(x, i^*, \left(\beta^{(1)}, \ldots, \beta^{(i-1)}\right)\right) \neq \mathcal{P}\left(x, i^*, (\beta_1, \ldots, \beta_{i-1})\right)$.

Definition 4.2 (ε-unambiguous $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IP)**.** *An $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ interactive protocol $(\mathcal{P}, \mathcal{V})$, in which we call $\mathcal{P}$ the* prescribed prover*, is an ε-*unambiguous *$(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IP for $\mathcal{L}$ if the following two properties hold:*

- **Prescribed Completeness:** *For every $x \in \{0,1\}^*$, if $\mathcal{V}$ interacts with $\mathcal{P}$ on common input x, then $\mathcal{V}$ outputs $\mathcal{L}(x)$ with probability 1.*
- **ε-Unambiguity:** *For every $x \in \{0,1\}^*$, every (computationally unbounded) cheating prover strategy $\widetilde{\mathcal{P}}$, every round $i^* \in [\ell]$, and for every $\beta^{(1)}, \ldots, \beta^{(i^*-1)}$, if $\widetilde{\mathcal{P}}$ first deviates from the protocol in round i^* (given the messages $\beta^{(1)}, \ldots, \beta^{(i^*-1)}$ in rounds $1, \ldots, (i^*-1)$ respectively), then at the end of the protocol $\mathcal{V}$ outputs a special reject symbol $\perp$ with probability $1 - \varepsilon(|x|)$, where the probability is over $\mathcal{V}$'s coin tosses in rounds $i^*, \ldots, \ell$.*

Note that in the unambiguity condition, the probability that $\mathcal{V}$ rejects is only over its random coin tosses in rounds $i^*, \ldots, \ell$.

The following proposition shows that *unambiguous interactive proofs* are in fact *interactive proofs* (as in Definition 4.1).

Proposition 4.3. *If $(\mathcal{P}, \mathcal{V})$ is an ε-unambiguous $(\ell, a, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IP for $\mathcal{L}$, then $(\mathcal{P}, \mathcal{V})$ is also an ε-sound $(\ell, a, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IP for $\mathcal{L}$.*

Proof. Completeness is trivial. For soundness, note that if $x \notin \mathcal{L}$, and a cheating prover $\widetilde{\mathcal{P}}$ does not deviate from the protocol $(\mathcal{P}, \mathcal{V})$, then $\mathcal{V}$ outputs $\mathcal{L}(x) = 0$ with probability 1 (by the prescribed completeness of $(\mathcal{P}, \mathcal{V})$). On the other hand, if $\widetilde{\mathcal{P}}$ does deviate at any point, then, by the unambiguity property, $\mathcal{V}$ rejects with probability $1 - \varepsilon$. □

Remark 4.4 (Reducing the Unambiguity Error)**.** *Recall that the soundness error of an interactive proof can be reduced (at an exponential rate) by repeating the protocol (either sequentially or in parallel). In contrast, neither sequential nor parallel repetition reduces the* unambiguity error *of an unambiguous interactive proof, since the deviating prover may deviate on only one of the copies of the base protocol.*

Although we will not use this fact, we remark that the unambiguity error can *be reduced at a cost that is roughly exponential in the number of rounds.*

4.3 Probabilistically Checkable Interactive Proofs (PCIPs)

Loosely speaking, Probabilistically Checkable Interactive Proofs[12] (PCIPs) are public-coin interactive protocols in which the verifier only queries the input and transcript at few locations (see Section 2.2.1 for further motivation). We view a PCIP as a two-step process: first, an interactive protocol is executed, where the verifier sends messages (which are merely random strings) to the prover, and receives in return messages from the prover. In the second step, the verifier queries just a few points in the transcript and input (without any further interaction with the prover), and decides whether to accept or reject. When referring to the verifier's running time we will refer only to the running time in the second step. In particular, the running time will typically be sub-linear in the transcript length (which is obviously impossible if we counted also the first step).

Definition 4.5 (ε-sound $(q_T, q_I, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP)**.** *A* public-coin $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IP *is an* ε-sound $(q_I, q_T, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ Probabilistically Checkable Interactive Proof (PCIP) for $\mathcal{L}$ *if the interaction consists of the following three phases:*

1. **Communication Phase:** *First, the two parties interact for ℓ rounds, in which $\mathcal{V}$ only sends random strings (of length b). No further interaction takes place after this phase.*
2. **Query Phase:** *$\mathcal{V}$ makes adaptive queries to its input and output. The number of queries to the transcript (resp., input) is at most $q_T = q_T(n)$ (resp., $q_I = q_I(n)$).*
3. **Decision Phase:** *Based on the answers to its queries and the random messages that it sent in the communication phase, $\mathcal{V}$ decides whether to accept or reject.*

In contrast to interactive protocols and proofs, here $\mathcal{V}\mathsf{time}$ *refers to $\mathcal{V}$'s running time only in the* query *and* decision *phases.*

Remark 4.6. *In Definition 4.5 we bound the verifier's queries both to the transcript* and to the input*. It is natural for the verifier to read its entire input, making a linear number of queries and achieving $q_I = n$, whereas the number of queries to the transcript is typically sublinear. For example, this is the case in classical PCP proofs for* NP *(which can be viewed as single-message* PCIP*s).*

In Section 4.3.1 we consider the case of sublinear PCIP *verifiers (under relaxed notions of soundness). In this setting, which will be our focus in much of this work, the numbers of queries to the transcript and input are both sublinear. Often, we do not need to distinguish between the two bounds. For convenience, we use the notation $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP to refer to a* PCIP *where the nubmers of verifier queries to the transcript and to the input are both bounded by q (i.e., a $(q, q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP as per Definition 4.5).*

Input-Oblivious PCIPs. The PCIPs that we construct will have a restricted type of query access to their transcript (and we will leverage this query access in our proof). Intuitively, we would like the verifier's queries to the transcript to depend only on its random string (and not on previous queries to the input and transcript). Unfortunately, not all of our protocols satisfy this requirement and so we settle for a relaxation that says that this is the case when interacting with the *prescribed prover*.

[12] An equivalent notion to PCIPs, called *interactive oracle proofs*, was introduced in an independent work of Ben-Sasson *et al.* [BCS16].

Definition 4.7. *We say that a* PCIP $(\mathcal{P}, \mathcal{V})$ *makes* input-oblivious queries *if for every two inputs* $x_1, x_2 \in \{0,1\}^n$ *and every random string* ρ*, the queries that the verifier* $\mathcal{V}$ *makes to the transcript when interacting with the prescribed prover* $\mathcal{P}$*, on input* x_1*, and* x_2*, both with the random string* ρ*, are the same.*

4.3.1 Sub-linear Time PCIPs

A natural extension of interactive and probabilistically checkable proofs, which has been studied in the literature [BFLS91, EKR04, BGH+06, DR06, RVW13], aims at obtaining *sub-linear* verification time. Since the verifier does not even have time to read its own input, we cannot hope to achieve soundness in general. Instead, we consider two relaxations:

Holographic Access to Input. If the input is encoded under an error-correcting code, it is often possible for the verifier to read only a sub-linear portion of the input. Indeed, it is known in the PCP literature [BFLS91] that query access to the low degree extension of the input suffices for sub-linear running time in the PCP setting, and the same is true also for PCIPs. Following [BFLS91] we refer to this type of access to the input as *holographic*.

Definition 4.8 (Holographic PCIP)**.** *Let* $\mathbb{H}$ *and* $\mathbb{F}$ *be constructible finite ensembles such that* $\mathbb{F}$ *is an extension field of* $\mathbb{H}$*. A* Holographic ε-sound $(q_I, q_T, \ell, a, b, \mathcal{P}\text{time}, \mathcal{V}\text{time})$ Probabilistically Checkable Interactive Proof (PCIP) for $\mathcal{L}$ *with* $(\mathbb{H}, \mathbb{F})$*-encoded input, is defined similarly to a* $(q_I, q_T, \ell, a, b, \mathcal{P}\text{time}, \mathcal{V}\text{time})$-PCIP *(see Definition 4.5), except that the verifier has oracle access to the low degree extension* $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$ *of the input* x*. The parameter* q_I *corresponds to the number of queries made to* $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$*.*

Similarly to Remark 4.6, in this work we often do not distinguish between the numbers of queries to the transcript and to the input, using a single parameter q to bound both quantities.

PCIP of Proximity. Inspired by the property testing [GGR98, EKR04] literature, another setting in which it is reasonable to consider a sub-linear number of queries to the input, is if we relax soundness and only require that the verifier reject inputs that are *far* from the language.

This type of extension has been previously considered in the PCP setting [BGH+06, DR06], the interactive proof setting [RVW13], and the NP (or rather MA) setting [GR15]. Here we consider the natural extension of PCIPs in this setting, which we call PCIP*s of proximity* (PCIPPs).

Definition 4.9 (PCIP of Proximity)**.** *An* ε*-sound* $(q_I, q_T, \ell, a, b, \mathcal{P}\text{time}, \mathcal{V}\text{time})$-Probabilistically Checkable Proof of Proximity (PCIPP) $(\mathcal{P}, \mathcal{V})$ *of* δ*-proximity for a language* $\mathcal{L}$*, where* $\delta = \delta(n) \in (0,1)$ *is called the* proximity parameter *and* $\varepsilon = \varepsilon(n) \in (0,1)$ *is the* soundness error*, is defined similarly to a* ε*-sound* $(q_I, q_T, \ell, a, b, \mathcal{P}\text{time}, \mathcal{V}\text{time})$-PCIP *(see Definition 4.5) except that the soundness is replaced with the following condition:*

- **(ε,δ)-Soundness:** *For every* x *that is* δ*-far from* $\mathcal{L}$ *and every cheating prover strategy* $\widetilde{\mathcal{P}}$*, the verifier* $\mathcal{V}$ *accepts when interacting with* $\tilde{P}$ *with probability less than* $\varepsilon(|x|)$*.*

4.3.2 PCIP for Pair Languages.

Following Ben-Sasson *et al.* [BGH+06] we consider pair languages which are languages whose input is divided into two parts. We typically think of an input (w, x) to a pair language $\mathcal{L}$ as being

composed of an explicit input w (to which the verifier has explicit access) and an implicit input x (to which the verifier only has implicit access). An ε-sound $(q_I, q_T, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP for a *pair language* $\mathcal{L}$ is defined similarly to Definition 4.5, except that both the verifier and the prover have explicit access to the explicit part of the input (and the completeness and soundness condition are modified to accommodate this). The definitions of Holographic PCIPs and PCIPPs are extended analogously, where for PCIPPs the (ε, δ)-soundness requirement applies to input pairs (w, x) such that x is δ-far from the set $\{x : (x, w) \in \mathcal{L}\}$.

We extend the notion of input-oblivious queries (see Definition 4.7) to PCIPs for pair languages but allow the verifier to use the explicit input to generate its queries. More specifically, a PCIP for a pair language $\mathcal{L}$ makes input-oblivious queries if for every two implicit input x_1 and x_2 and explicit input w, for every random string ρ the verifier makes the same queries to the transcript when interacting with the prescribed prover on input (w, x_1) and on input (w, x_2).

4.4 Unambiguous PCIPs w.r.t. Encoded Provers

The batch verification theorem that we prove does not hold for *every* PCIP, but rather for PCIPs that are, in a specific sense, unambiguous. At first glance, unambiguity might seem to be completely at odds with small query complexity: If a cheating prover $\widetilde{\mathcal{P}}$ changes just one bit of the i^{th} message, and the verifier only makes a small number of queries to the message, this change will likely go unnoticed, and unambiguity is lost. There are several different paths one could take towards reconciling these two notions, see Remark 4.10 below. We reconcile unambiguity with small query complexity by considering unambiguity for a restricted family of cheating provers. We design PCIPs where each message sent by the prescribed prover is a codeword in a high-distance error-correcting code (the low-degree extension). We restrict the cheating prover $\widetilde{\mathcal{P}}$ to only send messages that are themselves codewords. Thus, if the prover deviates in any round, it is sending an incorrect codeword, and must be deviating in many of the bits it sent. This can be detected even by making a small number of queries to that message.

We call a protocol that is unambiguous against this restricted class of cheating provers an unambiguous PCIP *w.r.t. encoded provers*. We note that this notion makes sense also in the PCP setting - a convenient way to describe classical PCPs (e.g., [BFLS91]) is to first construct a PCP under the relaxed assumption that the PCP must be encoded under the low degree extension encoding, and later to remove this assumption by using the low degree test and self correction property of polynomials (see Lemmas 3.3 to 3.5). Similarly, the assumption for PCIPs may be removed by using these two properties on the messages that the verifier receives.

In addition to the relaxation to so-called "encoded provers", we also assume that the verifier's input is encoded using a low-degree extension. This is similar to the definition of a holographic PCIP, see Definition 4.8, and to the "holographic proofs" of [BFLS91].

Remark 4.10 (Approximate unambiguity)**.** *A different path towards reconciling unambiguity with small query complexity would be relaxing to* approximate *unambiguity. In this variant, the cheating prover may deviate, but only in a δ-fraction of the bits of each message it sends.*[13] *If a cheating prover sends a message that's at distance δ or more from the prescribed message, then the verifier*

[13] Another option is to only require the verifier to reject with probability that is proportional to the "amount of unambiguity" (i.e., the distance of message on which the deviation occurred from the message that the prescribed prover would have sent). This corresponds to the notion of *strong* PCPs introduced by Goldreich and Sudan [GS06, Definition 5.6].

will end up rejecting with high probability. We note that unambiguity w.r.t. encoded provers can be used to derive approximate-unambiguity (using low-degree tests). In its own right, approximate-unambiguity may be a more natural notion than unambiguity w.r.t. encoded provers (since no restrictions are placed on a cheating prover). However, we find that the abstraction of unambiguity w.r.t. encoded provers more easily facilitates our constructions (and lets us avoid some amount of low-degree testing as we iteratively build our protocol). We do not use (or formally define) approximate unambiguity in this work.

Unambiguous PCIPs are formalized in Definition 4.12 below. We emphasize that this is a technical definition, which facilitates several of the transformations that we use in this work. We note that constructing such proof systems is indeed a relaxed goal: restricting the provers to a subset of all strategies makes soundness easier to achieve, and assuming that the input is encoded can (in several settings) dramatically reduce the verifier's query complexity into its input. Still, we show that unambiguous PCIPs w.r.t. encoded provers can be transformed into PCIPs and UIPs with the normal (more robust) notion of soundness and unambiguity. As always, we can do away with the assumption that the input is encoded by verifying all claims made about the input's low-degree extension. This requires adding a linear number of queries (to the input) and quasi-linear verification time. See Proposition 4.14. In the reverse direction, any unambiguous IP can be converted into an unambiguous PCIP w.r.t. encoded provers, see Proposition 4.15.

Parameter Setting. In order to formalize the above, we first need to set up the parameters for the low degree extension encoding. Fix constructible ensembles of finite fields (see Section 3.1) $\mathbb{H}$ and $\mathbb{F}$ such that $\mathbb{F}$ is an extension field of $\mathbb{H}$. Recall that the low-degree extension code $\mathsf{LDE}_{\mathbb{F},\mathbb{H}} : \{0,1\}^n \to \mathbb{F}^{\mathbb{F}^{\log_{|\mathbb{H}|}(n)}}$ (see Section 3.2) is an error-correcting code with distance $\left(1 - \frac{|\mathbb{H}| \cdot \log_{|\mathbb{H}|}(n)}{|\mathbb{F}|}\right)$.

Definition 4.11 (Encoded Prover)**.** *We say that a prover strategy $\mathcal{P}$ is $(g, \mathbb{H}, \mathbb{F})$-*encoded *if every message that $\mathcal{P}$ sends is composed of a sequence of $g = g(n)$ strings, each of which is an $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codeword, where $m = \log_{|\mathbb{F}|}(a/g)$, and a is the length of each prover message.*

Definition 4.12 (ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP w.r.t $(g, \mathbb{H}, \mathbb{F})$-encoded provers)**.** *We say that a* public-coin $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ *interactive protocol $(\mathcal{P}, \mathcal{V})$ is an* ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP for $\mathcal{L}$ w.r.t $(g, \mathbb{H}, \mathbb{F})$-encoded provers *if, for inputs of size n, the verifier $\mathcal{V}$ queries at most $q = q(n)$ symbols from the transcript and the low degree extension of the main input, and the following properties hold:*

1. *$\mathcal{P}$ is a $(g, \mathbb{H}, \mathbb{F})$-encoded prover strategy (see Definition 4.11).*
2. *$(\mathcal{P}, \mathcal{V})$ satisfy the prescribed completeness and ε-unambiguity conditions as in Definition 4.2, except that we only require ε-unambiguity against* $(g, \mathbb{H}, \mathbb{F})$-encoded *provers (rather than general cheating prover strategies).*
3. *Both for prescribed completeness and for unambiguity, we assume that the input is a codeword in $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m_{\mathsf{input}}}$, where $m_{\mathsf{input}} = \log_{|\mathbb{H}|}(n)$.*

Remark 4.13 (Variable Length unambiguous PCIPs)**.** *Definition 4.12 calls for protocols in which the prover's messages all have the same lengths. However, it will sometimes be more convenient for us to construct protocols in which the messages have varying lengths.*

Nevertheless, such protocols can be transformed into protocols with fixed length messages by having the prover pad messages to be sent with 0*'s* before encoding them. *Note that to preserve unambiguity the verifier must check this padding (since otherwise a deviation on the padded part would not be detected) and this can be done using the procedure in Proposition 3.9 (repeatedly if necessary). For a given security parameter $\lambda \geq 1$, this increases the unambiguity error of an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers to $\max(\varepsilon, 2^{-\lambda})$ and the number of queries (and verifier runtime) by $O(\ell \cdot g \cdot |\mathbb{H}| \cdot \log_{|\mathbb{H}|}(n) \cdot \lambda)$.*

Unambiguous PCIPs for Pair Languages. We extend the definition of unambiguous PCIPs (w.r.t. encoded provers) to *pair languages* as in Section 4.3. That is, the verifier is given explicit access to the explicit access of the input and implicit access to the low degree extension of the *implicit* input.

The following two propositions (Propositions 4.14 and 4.15) show that we can transform an unambiguous interactive proof into an unambiguous PCIP and vice versa (the former transformation is trivial whereas the latter requires some PCP machinery that is developed in Section 7).

Proposition 4.14 (From unambiguous PCIP w.r.t. encoded provers to unambiguous IP)**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$.*

If the language $\mathcal{L}$ has an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers, then $\mathcal{L}$ has an ε-unambiguous $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}')$-IP, where

$$\mathcal{V}\mathsf{time}' = \mathcal{V}\mathsf{time} + n \cdot \mathsf{poly}(|\mathbb{H}|, \log(|\mathbb{F}|), \log_{|\mathbb{H}|}(n)) + \ell \cdot a \cdot \mathsf{poly}(|\mathbb{F}|, \log_{|\mathbb{H}|}(a/g)).$$

Proof. Let $(\mathcal{P}, \mathcal{V})$ be an an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$ PCIP for $\mathcal{L}$ w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers. Consider a protocol $(\mathcal{P}', \mathcal{V}')$ in which $\mathcal{P}'$ just emulates $\mathcal{P}$ and $\mathcal{V}'$ first computes the low degree extension $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$ of the main input x and emulates $(\mathcal{P}, \mathcal{V})$ w.r.t. that low degree extension. In addition to $\mathcal{V}$'s checks, the verifier $\mathcal{V}'$ checks that each message $\alpha \in \mathbb{F}^a$ that it receives is composed of g $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords.

Prescribed completeness is trivial. For ε-unambiguity, observe that if a cheating prover sends a string that is not composed of g $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords, then $\mathcal{V}'$ immediately rejects and otherwise the cheating prover behaves like a g-encoded prover and so $\mathcal{V}'$ rejects with probability $1 - \varepsilon$.

All the parameters of $(\mathcal{P}', \mathcal{V}')$ are the same as in $(\mathcal{P}, \mathcal{V})$ except for the verifier's running time. The latter increases by an additive factor of $n \cdot \mathsf{poly}(|\mathbb{H}|, \log_{|\mathbb{H}|}(n)) + \ell \cdot a \cdot \mathsf{poly}(|\mathbb{F}|, m)$ due to computing the low degree extension of the input (see Proposition 3.7) and the additional checks.[14] □

Proposition 4.15 (From unambiguous IP to unambiguous PCIP w.r.t. encoded provers)**.** *Suppose that the language $\mathcal{L}$ has an ε-unambiguous $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IP $(\mathcal{P}, \mathcal{V})$ over the alphabet $\{0, 1\}$. We assume that $\mathcal{V}\mathsf{time} \geq \ell \cdot (a + b)$.*

Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$ and $|\mathbb{F}| = \mathsf{poly}(|\mathbb{H}|)$. Let $\sigma = \sigma(n) \in (0, 1)$ be a parameter. We assume that $\log(\max(n, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})) \leq |\mathbb{H}| \leq \mathsf{poly}(n, \ell, a)^{\sigma}$.

Then, $\mathcal{L}$ has an ε'-unambiguous $(q', \ell', a', b', \mathcal{P}\mathsf{time}', \mathcal{V}\mathsf{time}')$ PCIP $(\mathcal{P}', \mathcal{V}')$ over the alphabet $\mathbb{F}$ w.r.t. $(1, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, with the following parameters:

[14] To check that $f \in \mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ it suffices to check that for each one of the $m \cdot |\mathbb{F}|^{m-1}$ axis parallel lines, the restriction of f to that line is a degree $|\mathbb{H}| - 1$ individual degree polynomial. Thus, membership in the code $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ can be checked in time $m \cdot |\mathbb{F}|^{m-1} \cdot \mathsf{poly}(|\mathbb{F}|, m)$, see [RS96, Section 3] for details.

- $q' = \left(\mathcal{V}\mathsf{time}^{\sigma} + O(\ell \cdot \log(|\mathbb{F}|)\right)$.
- $\ell' = \left(\ell + O(1/\sigma)\right)$.
- $a' = \left(\mathsf{poly}(a, \ell, b, \mathcal{V}\mathsf{time}, |\mathbb{H}|)\right)$.
- $b' = (\max(b, O(|\mathbb{H}| \cdot \log(|\mathbb{F}|))))$.
- $\mathcal{P}\mathsf{time}' = (\mathcal{P}\mathsf{time} + \mathsf{poly}(n, a, \ell, b, \mathcal{V}\mathsf{time}, |\mathbb{H}|))$.
- $\mathcal{V}\mathsf{time}' = (\mathcal{V}\mathsf{time}^{\sigma} + (\mathsf{poly}(\ell, b, |\mathbb{H}|)))$.
- $\epsilon' = \left(\epsilon + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}\right)$.

Proof. We first trivially transform $(\mathcal{P}, \mathcal{V})$ into an ε''-unambiguous $(q'', \ell'', a'', b'', \mathcal{P}\mathsf{time}'', \mathcal{V}\mathsf{time}'')$-PCIP $(\mathcal{P}'', \mathcal{V}'')$ w.r.t. $(1, |\mathbb{H}|, |\mathbb{F}|)$-encoded provers (with input-oblivious queries), where:

- $q'' = n + \ell \cdot a$.
- $\ell'' = \ell$.
- $a'' = \mathsf{poly}(a)$.
- $b'' = b$.
- $\mathcal{P}\mathsf{time}'' = \mathcal{P}\mathsf{time} + \ell \cdot \mathsf{poly}(a, |\mathbb{H}|)$.
- $\mathcal{V}\mathsf{time}'' = \mathcal{V}\mathsf{time} + \ell \cdot \mathsf{poly}(a)$.
- $\epsilon'' = \epsilon$.

The latter is obtained by modifying $(\mathcal{P}, \mathcal{V})$ into a PCIP (with input-oblivious queries) by having the prover send its messages encoded under the low degree extension encoding $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ and the verifier reads the entire input and transcript. Note that the resulting protocol $(\mathcal{P}'', \mathcal{V}'')$ has very high query complexity and verifier running time.

Proposition 4.15 follows by applying the query reduction transformation (Lemma 8.2), which is shown in Section 7, to $(\mathcal{P}'', \mathcal{V}'')$ with respect to $\rho = O(1/\sigma)$. □

Remark 4.16. *We note that an ε-unambiguous* PCIP *w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers can also be transformed into a* sound PCIP *(as in Definition 4.5), with a slight loss in parameters by having the verifier compute the low degree extension of the main input, run low degree tests (see Lemmas 3.4 and 3.5) on every codeword that it receives and use the code's self-correction procedure to read from the codewords sent by the prover (see Lemma 3.3).*

For soundness, observe that if a cheating prover for the PCIP *sends any message that is far from an "encoded message" (i.e. a message that consists of g codewords from the $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}$ code), then the verifier rejects with high probability when performing the low degree test on that message. Otherwise (i.e., the messages are close to valid codewords), since the verifier uses the self-correction procedure, we can essentially treat the messages as being valid encoded messages, which means that the prover is behaving like an encoded prover. Soundness now follows from the unambiguity of the underlying unambiguous* PCIP *w.r.t. encoded provers.*

5 Our Results

Our main result is an unambiguous interactive proof for languages computable by bounded space Turing machines. We denote by $\mathsf{DTISP}(T,S)$ the class of all languages accepted by a Turing machine in time T and space S.

Theorem 7 (Unambiguous Interactive Proofs for Bounded Space)**.** *Let $T = T(n)$ and $S = S(n)$ such that $n \leq T \leq \mathsf{exp}(n)$ and $\log(T) \leq S \leq \mathsf{poly}(n)$.*

Let $\mathcal{L} \in \mathsf{DTISP}(T,S)$ and let $\delta = \delta(n) \in (0,1/2)$ such that $\mathsf{poly}(1/\delta) \leq \log(T)$. Then, $\mathcal{L}$ has a public-coin unambiguous interactive proof with perfect completeness and unambiguity error $\frac{1}{\mathsf{polylog}(T)}$. The number of rounds is $(1/\delta)^{O(1/\delta)}$. The communication complexity is $T^{O(\delta)} \cdot \mathsf{poly}(S)$. The (prescribed) prover runs in time $T^{1+O(\delta)} \cdot \mathsf{poly}(S)$ time, and the verifier runs in time $\big(n \cdot \mathsf{polylog}(T) + T^{O(\delta)} \cdot \mathsf{poly}(S)\big)$.

If the verifier is given query access to a low-degree extension of the input, then its running time is reduced to $T^{O(\delta)} \cdot \mathsf{poly}(S)$.

Using Proposition 4.3 we can state this result in terms of standard interactive proofs.[15]

Corollary 8 (Interactive Proofs for Bounded Space)**.** *Let $T = T(n)$ and $S = S(n)$ such that $n \leq T \leq \mathsf{exp}(n)$ and $\log(T) \leq S \leq \mathsf{poly}(n)$.*

Let $\mathcal{L} \in \mathsf{DTISP}(T,S)$ and let $\delta = \delta(n) \in (0,1/2)$ such that $\mathsf{poly}(1/\delta) \leq \log(T)$. Then, $\mathcal{L}$ has a public-coin interactive proof with perfect completeness and soundness error $\frac{1}{2}$. The number of rounds is $(1/\delta)^{O(1/\delta)}$. The communication complexity is $T^{O(\delta)} \cdot \mathsf{poly}(S)$. The (prescribed) prover runs in time $T^{1+O(\delta)} \cdot \mathsf{poly}(S)$ time, and the verifier runs in time $\big(n \cdot \mathsf{polylog}(T) + T^{O(\delta)} \cdot \mathsf{poly}(S)\big)$.

If the verifier is given query access to a low-degree extension of the input, then its running time is reduced to $T^{O(\delta)} \cdot \mathsf{poly}(S)$.

Lastly, we explicitly state our results for the setting of parameters that is the focus of our work: constant-round interactive proofs for languages computable in polynomial time and bounded polynomial space.

Corollary 9 (Constant-Round Interactive Proofs for Polynomial Time and Fixed Polynomial Space)**.** *Let $\mathcal{L}$ be a language that can be computed in time $\mathsf{poly}(n)$ and space $S = S(n)$. Then, for every constant $\delta > 0$, the language $\mathcal{L}$ has a constant-round public-coin interactive proof with perfect completeness and soundness error $\frac{1}{2}$. The communication complexity is $n^{\delta} \cdot \mathsf{poly}(S)$. The prover runs in time $\mathsf{poly}(n)$ and the verifier runs in time $\tilde{O}(n) + n^{\delta} \cdot \mathsf{poly}(S)$.*

If the verifier is given query access to a low-degree extension of the input, then its running time is reduced to $\mathsf{poly}(S) \cdot n^{\delta}$.

Remark 5.1 (Tightness of main result)**.** *First, we note that any language $\mathcal{L}$ that has an interactive proof with communication S and a verifier-space S, can also be computed in $O(S)$-space (though not necessarily in polynomial time). Thus, the communication's dependence on S seems inherent.*

Moreover, under reasonable complexity-theoretic conjectures, the trade-off between the number of rounds and the communication length is also (to some extent) tight. Suppose that a language $\mathcal{L}$ has an ℓ-round public-coin interactive proof, where the communication is bounded by c, and the

[15] We state our result for standard interactive proofs with soundness error $\frac{1}{2}$ but note that the soundness error can be reduced at an exponential rate using parallel repetition (in contrast, repetition does not decrease the *unambiguity* error, see Remark 4.4).

verifier runs in time $\tilde{O}(n)$. By collapsing the rounds a la [BM88], we obtain a two-message AM *protocol for $\mathcal{L}$, where the communication and verification time are increased by a multiplicative factor of $c^{\ell'}$, where ℓ' is an exponential function of ℓ. If we could get a constant number of rounds and $c = n^{o(1)}$, we would get a two-message* AM *protocol for $\mathcal{L}$ with $n^{o(1)}$ communication and $n^{1+o(1)}$-time verification.*

Thus, eliminating the tradeoff in Theorem 1 would imply a general (two-message) AM*-speedup theorem for bounded-space computations. Under the (seemingly reasonable) conjecture that no such general speedup exists, the theorem is tight. We note that there* is *hope for improvement in the dependence of the number of rounds on the constant δ.*

5.1 Roadmap for the Proof of Theorem 7

The main components in the proof of Theorem 7 are two transformations that we apply to unambiguous PCIPs (w.r.t. encoded provers). In Section 6 we show the first transformation, *batch verification* of PCIPs, which takes an unambiguous PCIP for a language $\mathcal{L}$ and generates an efficient unambiguous PCIP for checking membership of multiple inputs in $\mathcal{L}$. In Section 7 we show a complementary transformation that allows us to reduce the number of queries and verifier running time. In Section 8 we use these two transformation to prove Theorem 7. Lastly, in Section 9 we show how to batch verify general unambiguous interactive proofs rather than just unambiguous PCIPs (this last step is not required for the proof of Theorem 7, but we find this batch verification result to be of independent interest).

6 Batch Verification of Unambiguous PCIPs

In this section we show how to batch verify unambiguous PCIPs (w.r.t. encoded provers), which is the main component in our interactive-proofs for bounded-space computations. See Section 2 for an overview of the construction (we assume familiarity with this overview).

For a language $\mathcal{L}$, we denote by $\mathcal{L}^{\otimes k}$ the language

$$\mathcal{L}^{\otimes k} \stackrel{\text{def}}{=} \Big\{(x_1,\ldots,x_k) \,:\, \forall j \in [k],\ x_j \in \mathcal{L} \text{ and } |x_1| = \ldots = |x_k|\Big\}.$$

Our goal is to transform a PCIP for $\mathcal{L}$ into an efficient PCIP for $\mathcal{L}^{\otimes k}$.

As described in the technical overview, the construction has a recursive structure. To facilitate the recursion, we prove a more general batch verification lemma for *pair languages*. Recall that a pair language $\mathcal{L} \subseteq \{0,1\}^* \times \{0,1\}^*$, consists of pairs of the form (w,x), where view w as an explicit input and x as an implicit input. We extend the definition of $\mathcal{L}^{\otimes k}$ to *pair languages* in the following (slightly unnatural but technically convenient) way. For a pair language $\mathcal{L}$, we define

$$\mathcal{L}^{\otimes k} = \Big\{\,(w,(x_1,\ldots,x_k)) \,:\, \forall i \in [k],\ (w,x_i) \in \mathcal{L} \text{ and } |x_1| = \ldots = |x_k|\Big\}.$$

That is, the *same* explicit input is shared for all statements. In fact, it may be useful to think of the explicit input w as a common reference string shared among the k interactive protocols to be batched.[16]

[16] Actually, in our use of Lemma 6.1 the explicit input w will always be the empty string. However, the recursive step will generate an explicit input and so we add it here for convenience.

Lemma 6.1 (Batch Verification for unambiguous PCIPs w.r.t. encoded provers (Pair Languages), see also Lemma 8.1)**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$ and $|\mathbb{F}| \leq \mathsf{poly}(|\mathbb{H}|)$.*

Let $(\mathcal{P}, \mathcal{V})$ be an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP for the pair language $\mathcal{L}$, with respect to $(g, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries. Let $k = k(n) \geq 1$, $\mu = \mu(n) \geq 1$ and let $\lambda = \lambda(n) \geq 1$ be a security parameter, where:

- $\log(\ell^{\mu} \cdot a) \leq \min\left(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\right)$.
- $a \geq \mathsf{poly}(k, q, g) \cdot (\mathsf{poly}(\lambda, |\mathbb{H}|, \ell))^{\mu}$.
- $\mathcal{P}\mathsf{time} \geq \ell \cdot a \cdot \mathsf{polylog}(|\mathbb{F}|)$.

Then, there exists an ε_{A}-unambiguous $(q_{\mathsf{A}}, \ell_{\mathsf{A}}, a_{\mathsf{A}}, b_{\mathsf{A}}, \mathcal{P}\mathsf{time}_{\mathsf{A}}, \mathcal{V}\mathsf{time}_{\mathsf{A}})$-PCIP for the pair language $\mathcal{L}^{\otimes k}$ w.r.t. $(g_{\mathsf{A}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, with the following parameters:

- $\varepsilon_{\mathsf{A}} = (\mu + 1) \cdot \left(\varepsilon + 3 \cdot 2^{-\lambda}\right)$.
- $q_{\mathsf{A}} = k^2 \cdot q \cdot \mathsf{poly}(g) \cdot \left(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\right)^{\mu+1}$.
- $\ell_{\mathsf{A}} = \ell \cdot (\mu + 1) + 2\mu$.
- $b_{\mathsf{A}} = \max\left(b, k \cdot \mathsf{poly}(|\mathbb{H}|, \ell^{\mu}, g, \lambda)\right)$.
- $a_{\mathsf{A}} = a \cdot 2k^{1/\mu} \cdot \log(k) \cdot \lambda \cdot \ell^{\mu}$.
- $\mathcal{P}\mathsf{time}_{\mathsf{A}} = 2k \cdot \mu \cdot \ell^{\mu} \cdot \mathcal{P}\mathsf{time} + \mathcal{V}\mathsf{time} \cdot q \cdot k^2 \cdot \mathsf{poly}(g) \cdot \left(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\right)^{\mu+1}$.
- $\mathcal{V}\mathsf{time}_{\mathsf{A}} = \mathcal{V}\mathsf{time} \cdot q \cdot k^2 \cdot \mathsf{poly}(g) \cdot \left(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\right)^{\mu+1}$.
- $g_{\mathsf{A}} = g \cdot 2k^{1/\mu} \cdot \log(k) \cdot \lambda \cdot \ell^{\mu}$.

Most importantly, note that the length of $\mathcal{P}_{\mathsf{Batch}}$'s messages is roughly $a \cdot k^{1/\mu}$, rather than the trivial $a \cdot k$ (which can be obtained by simply running the k protocols).

Remark 6.2. *We remark that, given oracle access to the prover $\mathcal{P}$ for the base PCIP for $\mathcal{L}$ the prescribed prover for $\mathcal{L}^{\otimes k}$ in Lemma 6.1 can be implemented in time $\mathcal{P}\mathsf{time}_{\mathsf{A}} = \Big(O(a \cdot k \cdot \ell^{\mu} \cdot \mu \cdot \mathsf{polylog}(|\mathbb{F}|)) + \mathcal{V}\mathsf{time} \cdot k^2 \cdot q \cdot \mathsf{poly}(g) \cdot \left(\mathsf{poly}(\lambda, |\mathbb{H}|, \ell)\right)^{\mu+1}\Big)$. This observation is particularly useful in a setting in which $\mathcal{P}$ is given as input also a witness to the statement $x_j \in \mathcal{L}$ (e.g., when batching UP statements).*

As alluded to above, the proof of Lemma 6.1 is based on a recursive protocol. At each step the task of batching k base protocols will be reduced, via an interactive protocol, to that of batching $k' \ll k$ (slightly more complicated) protocols. At the very end, when we are left with sufficiently few protocols, the prover and verifier simply execute these protocols.

We begin with the recursive step: that is, how to (interactively) reduce the batching of k protocols to the batching of $k' \ll k$ protocols. Toward this end, we prove a lemma, which we call the "Deviation Amplification Lemma" (Lemma 6.3). This lemma shows an interactive protocol

that takes as input an explicit input w and k implicit inputs $x_1, \ldots, x_k$ for $\mathcal{L}$, and outputs a related explicit input w' and k related implicit inputs $x'_1, \ldots, x'_k$ for a slightly more complicated language $\mathcal{L}'$ such that:

1. If $\forall j \in [k],\ (w, x_j) \in \mathcal{L}$, then $\forall j \in [k],\ (w', x'_j) \in \mathcal{L}'$.
2. If $\exists j \in [k],\ (w, x_j) \notin \mathcal{L}$, then either the verifier rejects or there exists a set $J \subset [k]$ of size $|J| \geq d$ such that $\forall j \in J,\ (w', x'_j) \notin \mathcal{L}'$ (where $1 \ll d \ll k$ is a parameter that will be set below).

That is, the Deviation Amplification Lemma allows us to *amplify* the number of false statements. Lemma 6.1 follows by first applying the Deviation Amplification Lemma, then randomly sampling roughly k/d of the new statements and recursing on the sampled statements (the latter step is done in Section 6.3).

Organization of Section 6. In Section 6.1 we state the Deviation Amplification Lemma (Lemma 6.3). In Section 6.2 we prove this lemma and in Section 6.3 we use it to prove the batch verification lemma (Lemma 6.1). Lastly, Section 6.4 contains a couple of less central proofs that were deferred from Section 6.2.

6.1 The Deviation Amplification Lemma

Before stating the Deviation Amplification Lemma, we briefly go over its (many) parameters: k is the number of protocols to be batched, $d \in [k/\log(k)]$ is a parameter that gives a trade-off between the amount of communication in the Deviation Amplification Protocol and the number of false statements that it outputs, and λ is a security parameter (that can be used to decrease the unambiguity error). Finally, q, ℓ, b, a and g are, respectively, the query complexity, number of rounds, length of verifier messages, length of prover messages, and number of $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords per prover message, in the base protocol to be batched.

Lemma 6.3 (Deviation Amplification Lemma)**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$ and $|\mathbb{F}| \leq \mathsf{poly}(|\mathbb{H}|)$.*

Suppose that the pair language $\mathcal{L}$ has an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-$\mathsf{PCIP}$ $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries. Let $k = k(n) \geq 1$, $d = d(n) \in [k/\log(k)]$ and $\lambda = \lambda(n) \geq 1$ be parameters and assume that: (1) $\log(a) \leq \min\left(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\right)$, (2) $a \geq \mathsf{poly}(k, q, \ell, g, \lambda, |\mathbb{H}|)$, and (3) $\mathcal{P}\mathsf{time} \geq \ell \cdot a \cdot \mathsf{polylog}(|\mathbb{F}|)$. Then, there exists a pair language $\mathcal{L}'$ such that:

1. *$\mathcal{L}'$ has an ε'-unambiguous $(q', \ell', b', a', \mathcal{P}\mathsf{time}', \mathcal{V}\mathsf{time}')$-$\mathsf{PCIP}$ $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$ w.r.t. $(g', \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and the following parameters:*
   - $\varepsilon' = \varepsilon + 2^{-\lambda}$.
   - $q' = \left(q \cdot \left(\ell + \mathsf{poly}(\lambda, |\mathbb{H}|)\right) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\right)$.
   - $\ell' = \ell$.
   - $b' = b$.
   - $a' = \ell \cdot a$.

- $\mathcal{P}\mathsf{time}' = \ell \cdot \mathcal{P}\mathsf{time}$.
- $\mathcal{V}\mathsf{time}' = \Big(\ell \cdot \mathcal{V}\mathsf{time} + q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\Big)$.
- $g' = \ell \cdot g$.

2. *There exists an $(\ell+1)$-round interactive protocol $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$, such that the prover $\mathcal{P}_{\mathsf{amplify}}$, which is $(d \cdot \log(k) \cdot g, \mathbb{H}, \mathbb{F})$-encoded, is given as input a string $w \in \{0,1\}^{n_{\mathsf{explicit}}}$ and a sequence $(x_1, \ldots, x_k) \in (\{0,1\}^n)^k$, and the verifier $\mathcal{V}_{\mathsf{amplify}}$ is given explicit access to w and implicit access to $(x_1, \ldots, x_k)$ and makes input-oblivious queries. After the two parties interact, the verifier $\mathcal{V}_{\mathsf{amplify}}$ either rejects or outputs a string w' and a sequence $(x'_j)_{j \in [k]}$ such that:*

   - **Prescribed Completeness:** *If $\mathcal{V}_{\mathsf{amplify}}$ interacts with $\mathcal{P}_{\mathsf{amplify}}$, and $\forall j \in [k]$, $(w, x_j) \in \mathcal{L}$, then $\forall j \in [k]$, $\Big((w, w'), (x_j, x'_j)\Big) \in \mathcal{L}'$. On the other hand, if $\mathcal{V}_{\mathsf{amplify}}$ interacts with $\mathcal{P}_{\mathsf{amplify}}$ but $\exists j \in [k]$, $(w, x_j) \notin \mathcal{L}$, then $\mathcal{V}_{\mathsf{amplify}}$ rejects with probability 1.*
   - **Unambiguity:** *For every $(d \cdot \log(k) \cdot g, \mathbb{H}, \mathbb{F})$-encoded prover strategy $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ and every round $i^* \in [\ell+1]$, if $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ first deviates from the protocol at round i^*, then with probability $1-\varepsilon-2^{-\lambda}$ over the coin tosses of $\mathcal{V}_{\mathsf{amplify}}$ in rounds $(i^*, \ldots, \ell+1)$, either $\mathcal{V}_{\mathsf{amplify}}$ rejects or there exists a set $J \subset [k]$ of size $|J| = d/2$ such that $\forall j \in J$, $\Big((w, w'), (x_j, x'_j)\Big) \notin \mathcal{L}'$.*

   The protocol $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$ has the following parameters:

   (a) Query complexity: $\Big(q \cdot k \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k^2 \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\Big)$.

   (b) $\mathcal{V}_{\mathsf{amplify}}$ message length (in field elements) $\max\big(b, k \cdot \mathsf{poly}(|\mathbb{H}|, g, \ell, \lambda)\big)$.

   (c) $\mathcal{P}_{\mathsf{amplify}}$ message length (in field elements): $d \cdot \log(k) \cdot a$.

   (d) $\mathcal{V}_{\mathsf{amplify}}$ time: $\Big(k \cdot \mathcal{V}\mathsf{time} + q \cdot k \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k^2 \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\Big)$.

   (e) $\mathcal{P}_{\mathsf{amplify}}$ time: $\Big(2k \cdot \mathcal{P}\mathsf{time} + k \cdot \mathcal{V}\mathsf{time} + k \cdot q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k^2 \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\Big)$.

6.2 Proof of the Deviation Amplification Lemma (Lemma 6.3)

We start by defining the language $\mathcal{L}'$. Then, we show that it satisfies the conditions stated in Lemma 6.3 by first constructing the Deviation Amplification Protocol $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$ (this is the main step in the proof). The Deviation Amplification Protocol is specified in Section 6.2.1, its analysis is in Section 6.2.2. We then construct a PCIP protocol $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$ for $\mathcal{L}'$, see Section 6.2.3.

Fix $m \stackrel{\text{def}}{=} \log_{|\mathbb{F}|}(a/g)$. Recall that a $(g, \mathbb{H}, \mathbb{F})$-encoded strategy is a prover strategy in which the cheating prover is only allowed to send messages that are composed of g $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords (note that the length of such messages over the alphabet $\mathbb{F}$ is $g \cdot |\mathbb{F}|^m = a$). Also note that $m \leq \log(a) \leq |\mathbb{H}|$ (by our assumption that $\log(a) \leq |\mathbb{H}|$).

The Language $\mathcal{L}'$. The pair language $\mathcal{L}'$ is an "augmented" version of $\mathcal{L}$. We view inputs to $\mathcal{L}'$ as being composed of two parts: an explicit input and an implicit input. The explicit input for $\mathcal{L}'$ consists of three parts:

1. An explicit input $w \in \{0,1\}^{n_{\mathsf{explicit}}}$ for $\mathcal{L}$.
2. A sequence of verifier messages $\beta^{(1)}, \ldots, \beta^{(\ell)} \in \mathbb{F}^b$ for the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$.
3. A sequence of indices $S \subseteq [\ell \cdot a]$ of size $|S| = q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)$. We think of S as specifying coordinates in the prover's messages in rounds $1, \ldots, \ell$, respectively.

The implicit input for $\mathcal{L}'$ consists of two parts:

1. An implicit input $x \in \{0,1\}^n$ for $\mathcal{L}$.
2. A sequence of values $\phi : S \to \mathbb{F}$. We think of ϕ as specifying values for the prover's messages for the coordinates in S. (ϕ is represented as a string of length $O(|S| \cdot \log(|\mathbb{F}|))$ in the natural way).

Loosely speaking, the requirement is that $(w, x) \in \mathcal{L}$ and that the messages sent by the prescribed prover in the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ (on input x with verifier messages $\left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)$) projected to the set of indices S, are equal to the corresponding values ϕ.

Definition 6.4 (The Language $\mathcal{L}'$)**.** *The pair language $\mathcal{L}'$ consists of all tuples*

$$\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right), S\right), (x, \phi)\right),$$

where $w \in \{0,1\}^{n_{\mathsf{explicit}}}$, $\beta^{(1)}, \ldots, \beta^{(\ell)} \in \mathbb{F}^b$, $S \subseteq [\ell \cdot a]$ of size $|S| = q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)$, $x \in \{0,1\}^n$, and $\phi : S \to \mathbb{F}$ such that

1. *$(w, x) \in \mathcal{L}$; and*
2. *$\forall \xi \in S$, $\left(\alpha^{(1)}, \ldots, \alpha^{(\ell)}\right)\big|_{\xi} = \phi(\xi)$,*

where $\forall i \in [\ell]$, $\alpha^{(i)} \stackrel{\text{def}}{=} \mathcal{P}_{\mathcal{L}}\left((x, w), i, \left(\beta^{(1)}, \ldots, \beta^{(i-1)}\right)\right)$.

(Recall that $x|_{\xi}$ denotes the ξ^{th} entry in x, see Section 3).

In order to present the Deviation Amplification Protocol, we require two additional ingredients: (1) a robust procedure, which we call the *transcript tester*, for checking that a given transcript is close to an accepting transcript for the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ and (2) a checksum function ENC, which is a certain type of error correcting encoding that will be specified next.

Transcript Tester. The verifier in a PCIP w.r.t. encoded prover can be thought of as a sub-linear time algorithm for checking *encoded* transcripts. For our construction we will need a robust version of the verifier that rejects transcripts that are not encoded, as long as they are *far* from any encoded transcript. We construct such a procedure, which we call the *transcript tester* by utilizing the local testability and decodability of the low degree extension encoding.

Hence, the transcript tester $\mathcal{T}$ can be thought of as a *property tester* [GGR98] for checking proximity to an accepting transcript for $\mathcal{V}_{\mathcal{L}}$ (recall that $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ is the interactive proof for the language $\mathcal{L}$, which we want to batch). That is, $\mathcal{T}$ is given as explicit input the coins of the verifier $\mathcal{V}_{\mathcal{L}}$ and as implicit input the prover messages for the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$. The tester $\mathcal{T}$ accepts if the transcript is composed of codewords that make $\mathcal{V}_{\mathcal{L}}$ accept and rejects (with high probability) if the transcript is far from any transcript that would make $\mathcal{V}_{\mathcal{L}}$ accept. We emphasize that the latter condition should hold even for transcripts that are not $(g, \mathbb{H}, \mathbb{F})$-encoded.

Lemma 6.5 (Transcript Tester)**.** *For every proximity parameter $\delta \in (0, 1/3)$ and soundness parameter $\lambda \geq 1$, there exists a randomized algorithm $\mathcal{T}$, called the* transcript tester*, that is given as explicit input $w \in \{0,1\}^{n_{\text{explicit}}}$, $\beta^{(1)}, \ldots, \beta^{(\ell)} \in \mathbb{F}^b$ and as implicit input $\hat{x} = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$, where $x \in \{0,1\}^n$, and $\tilde{\alpha}^{(1)}, \ldots, \tilde{\alpha}^{(\ell)} \in \mathbb{F}^a$. For every $x \in \{0,1\}^n$, $w \in \{0,1\}^{n_{\text{explicit}}}$ and $\beta^{(1)}, \ldots, \beta^{(\ell)} \in \mathbb{F}^b$ it holds that:*

- **Completeness:** *If $\tilde{\alpha}^{(1)}, \ldots, \tilde{\alpha}^{(\ell)} \in \mathbb{F}^a$ are each composed of g codewords in $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$, then the tester $\mathcal{T}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(\hat{x}, \left(\tilde{\alpha}^{(1)}, \ldots, \tilde{\alpha}^{(\ell)}\right)\right)\right)$ accepts if and only if the verifier $\mathcal{V}_{\mathcal{L}}$ accepts $\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(x, \left(\tilde{\alpha}^{(1)}, \ldots, \tilde{\alpha}^{(\ell)}\right)\right)\right)$.*
- **Soundness:** *If $\left(\tilde{\alpha}^{(1)}, \ldots, \tilde{\alpha}^{(\ell)}\right)$ is δ-far from the set*

$$\left\{ \left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right) \in ((\mathsf{LDE}_{\mathbb{F},\mathbb{H},m})^g)^\ell \ : \ \mathcal{V}_{\mathcal{L}}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(\hat{x}, \left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right)\right)\right) = 1 \right\}$$

  then, with probability $1 - 2^{-\lambda}$, the tester $\mathcal{T}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(x, \left(\tilde{\alpha}^{(1)}, \ldots, \tilde{\alpha}^{(\ell)}\right)\right)\right)$ rejects.

The algorithm $\mathcal{T}$ runs in time $\mathcal{V}\mathsf{time} + q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + \frac{1}{\delta} \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)$, makes $q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + \frac{1}{\delta} \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)$ queries to its implicit input, and uses $\frac{1}{\delta} \cdot \mathsf{poly}(|\mathbb{H}|, g, \ell, \lambda)$ random bits.

Furthermore, the queries that $\mathcal{T}$ makes to the $\alpha^{(i)}$'s are input-oblivious: for every two inputs $x_1, x_2 \in \{0,1\}^n$, every $w \in \{0,1\}^{n_{\text{explicit}}}$, every $\beta^{(1)}, \ldots, \beta^{(\ell)} \in \mathbb{F}^b$ and every random string ρ, if $\alpha_1^{(i)} \stackrel{\text{def}}{=} \mathcal{P}_{\mathcal{L}}\left((x_1, w), i, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right)$ and $\alpha_2^{(i)} \stackrel{\text{def}}{=} \mathcal{P}_{\mathcal{L}}\left((x_2, w), i, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right)$ are the honestly generated messages by the prescribed prover*, then $\mathcal{T}$ makes the same queries to the $\alpha^{(i)}$'s when given as input either $\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(x_1, \left(\alpha_1^{(1)}, \ldots, \alpha_1^{(\ell)}\right)\right)\right)$ or $\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(x_2, \left(\alpha_2^{(1)}, \ldots, \alpha_2^{(\ell)}\right)\right)\right)$.*

The (fairly straightforward) proof of Lemma 6.5, which is based on the low degree test (see Lemmas 3.4 and 3.5) and the self-correction property of polynomials (see Lemma 3.3), is deferred to Section 6.4.1.

Checksum. Let $k \in \mathbb{N}$ and $d \leq \frac{k}{\log(k)}$. The following proposition shows the existence of a *systematic*[17] linear error-correcting code $C : \mathbb{F}^k \to \mathbb{F}^{k+d \cdot \log(k)}$ with absolute distance $d+1$ and $\mathsf{poly}(\log(|\mathbb{F}|), k)$ time encoding.[18] We will use this code as a checksum function.

Proposition 6.6. *Let $\mathbb{F}$ be a constructible field ensemble (see Definition 3.2) and let $k = k(n) \geq 1$ and $d = d(n) \leq k/\log(k)$. There exists a* systematic *$\mathbb{F}$-linear error-correcting code $C : \mathbb{F}^k \to \mathbb{F}^{k+d \cdot \log(k)}$ with absolute distance $d+1$ and $k \cdot \mathsf{poly}(\log(|\mathbb{F}|), \log(k))$-time encoding.*

The classical Reed-Solomon code almost satisfies the requirements in Proposition 6.6 (even with slightly better rate). The problem however, is that it requires the field size to be quite large (i.e., $|\mathbb{F}| \gg k$), which turns out to be problematic. Since the actual construction (which is a variant of the Reed Solomon code) is not important for the rest of the proof of Lemma 6.3, we defer the proof of Proposition 6.6 to Section 6.4.2.

Since C is systematic, there exists a function $\mathsf{ENC} : \mathbb{F}^k \to \mathbb{F}^d$ such that $C(\eta) \equiv (\eta, \mathsf{ENC}(\eta))$. We will often refer to $chksum = \mathsf{ENC}(\eta)$ as the checksum of the vector η.

[17] Recall that an error-correcting code C is systematic if there exists a function ENC such that $C(\eta) \equiv (\eta, \mathsf{ENC}(\eta))$.

[18] We remark that the Reed-Solomon code over $\mathbb{F}$ satisfies the above but requires a field of size $|\mathbb{F}| \geq k$ and so we use a different construction.

We define a function $\mathsf{DEC} : \mathbb{F}^{k+d\cdot\log(k)} \to \mathbb{F}^k$ to be the minimum distance decoding function for C that *preserves checksums*. That is, on input $\eta \in \mathbb{F}^k$ and $chksum \in \mathbb{F}^{d\log(k)}$, the function DEC outputs the vector $\eta' \in \mathbb{F}^k$ that is the closest vector to η with $\mathsf{ENC}(\eta') = chksum$, where ties are broken in some consistent canonical way.

We extend ENC and DEC to operate over matrices (rather than just vectors) by operating column-by-column. That is, $\mathsf{ENC} : \mathbb{F}^{k\times m} \to \mathbb{F}^{(d\cdot\log(k))\times m}$ is defined as $\mathsf{ENC}(\eta_1,\ldots,\eta_m) \equiv \big(\mathsf{ENC}(\eta_1),\ldots,\mathsf{ENC}(\eta_m)\big)$. Similarly, we define $\mathsf{DEC} : \mathbb{F}^{k\times m} \times \mathbb{F}^{d\cdot\log(k)\times m} \to \mathbb{F}^{k\times m}$ as

$$\mathsf{DEC}\Big((\eta_1,\ldots,\eta_m),\big(chksum_1,\ldots,chksum_m\big)\Big) \equiv \Big(\mathsf{DEC}(\eta_1,chksum_1),\ldots,\mathsf{DEC}(\eta_m,chksum_m)\Big).$$

6.2.1 The Deviation Amplification Protocol

The Deviation Amplification Protocol $(\mathcal{P}_{\mathsf{amplify}},\mathcal{V}_{\mathsf{amplify}})$ is presented in Fig. 1.

6.2.2 Completeness, Unambiguity and Complexity of the Protocol $(\mathcal{P}_{\mathsf{amplify}},\mathcal{V}_{\mathsf{amplify}})$

Prescribed Completeness of $(\mathcal{P}_{\mathsf{amplify}},\mathcal{V}_{\mathsf{amplify}})$. Prescribed completeness follows directly from the prescribed completeness of $(\mathcal{P}_{\mathcal{L}},\mathcal{V}_{\mathcal{L}})$ and the completeness of $\mathcal{T}$, details follow.

Let $x_1,\ldots,x_k \in \{0,1\}^n$ and $w \in \{0,1\}^{n_{\mathsf{explicit}}}$. By the prescribed completeness of $(\mathcal{P}_{\mathcal{L}},\mathcal{V}_{\mathcal{L}})$, for every $j \in [k]$, it holds that

$$\mathcal{V}_{\mathcal{L}}\left(\left(w,\left(\beta^{(1)},\ldots,\beta^{(\ell)}\right)\right),\left(\hat{x}_j,\left(\alpha_j^{(1)},\ldots,\alpha_j^{(\ell)}\right)\right)\right) = \mathcal{L}(x_j) \tag{2}$$

with probability 1 over the choice of $\beta^{(1)},\ldots,\beta^{(\ell)}$, where $\hat{x}_j = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_j)$ and $\forall i \in [\ell]$, $\alpha_j^{(i)} \stackrel{\text{def}}{=} \mathcal{P}_{\mathcal{L}}\left(x_j,i,\left(\beta^{(1)},\ldots,\beta^{(i-1)}\right)\right)$. Since each $\alpha_j^{(i)}$ is indeed composed of g $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords, by Eq. (2) and the completeness of $\mathcal{T}$,

$$\mathcal{T}\left(\left(w,\left(\beta^{(1)},\ldots,\beta^{(\ell)}\right)\right),\left(\hat{x}_j,\left(\tilde{\alpha}^{(1)},\ldots,\tilde{\alpha}^{(\ell)}\right)\right)\right) = \mathcal{L}(x_j). \tag{3}$$

Therefore, since $\mathcal{P}_{\mathsf{amplify}}$'s answers in Phase II (Step 3) are consistent with the $\alpha_j^{(i)}$'s, if $\exists j \in [k]$, $(w,x_j) \notin \mathcal{L}$, then $\mathcal{T}$ rejects and so V_{amplify} rejects.

Assume that $\forall j \in [k]$, $(w,x_j) \in \mathcal{L}$. By construction, $\forall \xi \in S$, $\phi(\xi) = \mathbf{A}|_\xi$, where $\mathbf{A} = \big(A^{(1)},\ldots,A^{(\ell)}\big)$, $A^{(i)} \in \mathbb{F}^{k\times a}$ is a matrix whose j^{th} row is $\alpha_j^{(i)}$ and $chksum^{(i)} = \mathsf{ENC}(A^{(i)})$ for every $i \in [\ell]$.

For every $\xi \in S$ it holds that $\mathsf{ENC}(\phi(\xi)) = \mathsf{ENC}(\mathbf{A}|_\xi) = \mathbf{chksum}|_\xi$, where $\mathbf{chksum} = \big(chksum^{(1)},\ldots,chksum^{(\ell)}\big)$ and $chksum^{(i)} = \mathsf{ENC}(A^{(i)})$ for every $i \in [\ell]$. Hence, $\mathcal{V}_{\mathsf{amplify}}$'s consistency check passes.

Completeness follows by observing that, by definition of $\mathcal{L}'$, for every $j \in [k]$ it holds that $\big(\big(w,\big(\beta^{(1)},\ldots,\beta^{(\ell)}\big),S\big),(x_j,\phi_j)\big) \in \mathcal{L}'$.

We also note that since $\mathcal{T}$ makes input-oblivious queries (see Lemma 6.5), the queries that $\mathcal{V}_{\mathsf{amplify}}$ makes when interacting with $\mathcal{P}_{\mathcal{L}}$ (which are essentially the queries generated by $\mathcal{T}$ for all k inputs) are also input-oblivious.

The Deviation Amplification Protocol $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$

Prover Input: $x_1, \ldots, x_k \in \{0,1\}^n$ and $w \in \{0,1\}^{n_{\mathsf{explicit}}}$.
Verifier Input: Explicit access to w and implicit access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$.[a]

(Phase I)

1. For $i = 1, \ldots, \ell$:
   (at the beginning of round i the prover $\mathcal{P}_{\mathsf{amplify}}$ already knows $\beta^{(1)}, \ldots, \beta^{(i-1)}$)
   (a) $\forall j \in [k]$, $\mathcal{P}_{\mathsf{amplify}}$ computes the message $\alpha_j^{(i)} = \mathcal{P}_{\mathcal{L}}\left(x_j, i, \left(\beta^{(1)}, \ldots, \beta^{(i-1)}\right)\right) \in \mathbb{F}^a$.
   (b) $\mathcal{P}_{\mathsf{amplify}}$ constructs a matrix $A^{(i)} \in \mathbb{F}^{k \times a}$ whose j^{th} row is $\alpha_j^{(i)}$, for every $j \in [k]$.
   $\mathcal{P}_{\mathsf{amplify}}$ computes and sends $chksum^{(i)} = \mathsf{ENC}\left(A^{(i)}\right) \in \mathbb{F}^{d \cdot \log(k) \times a}$ to $\mathcal{V}_{\mathsf{amplify}}$,
   which receives a matrix $\widetilde{chksum}^{(i)} \in \mathbb{F}^{d \cdot \log(k) \times a}$ (which is allegedly equal to $chksum^{(i)}$).
   (c) $\mathcal{V}_{\mathsf{amplify}}$ chooses uniformly at random $\beta^{(i)} \in \mathbb{F}^b$ and sends $\beta^{(i)}$ to $\mathcal{P}_{\mathsf{amplify}}$.

(Phase II)

2. $\mathcal{V}_{\mathsf{amplify}}$ generates and sends to $\mathcal{P}_{\mathsf{amplify}}$ a random string ρ for the transcript tester $\mathcal{T}$ (with respect to proximity parameter $\frac{1}{2kg\ell}$ and soundness parameter λ).

3. For every $j \in [k]$, the prover $\mathcal{P}_{\mathsf{amplify}}$ generates the set of queries S_j that $\mathcal{T}$ makes given explicit input $\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right)$ and implicit input $\left(x_j, \left(\alpha_j^{(1)}, \ldots, \alpha_j^{(\ell)}\right)\right)$. Since $\mathcal{T}$ makes *input-oblivious queries* (see Lemma 6.5), it holds that $S_1 = \cdots = S_k$ and so we denote $S \stackrel{\text{def}}{=} S_1$.

   For every $j \in [k]$, let $\phi_j : S \to \mathbb{F}$ be defined as $\phi_j(\xi) \stackrel{\text{def}}{=} \left(\alpha_j^{(1)}, \ldots, \alpha_j^{(\ell)}\right)\Big|_{\xi}$.

   The prover $\mathcal{P}_{\mathsf{amplify}}$ sends (S, ϕ) to $\mathcal{V}_{\mathsf{amplify}}$ encoded as an $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codeword by padding the message with 0's (by our assumption on the parameters there is sufficient room for ϕ to be encoded).

4. $\mathcal{V}_{\mathsf{amplify}}$ gets (the low degree extensions of) $\tilde{S}$ and $\tilde{\phi}_1, \ldots, \tilde{\phi}_k : \tilde{S} \to \mathbb{F}$ (which are allegedly S and $\phi_1, \ldots, \phi_k$), checks the padding (see Remark 4.13) and checks that:
   (a) **Transcript Test:** For every $j \in [k]$, the transcript tester $\mathcal{T}$ accepts when given as explicit input $\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right)$, the random coins ρ, proximity parameter $\frac{1}{2kg\ell}$, soundness parameter λ. The input queries that $\mathcal{T}$ makes are answered based on $\mathsf{LDE}_{\mathbb{F},H}(x_j)$ (which can be accessed using a single query to the main input, see Proposition 3.8) and each transcript query, which is some $\xi \in \tilde{S}$, is answered with $\tilde{\phi}_j(\xi)$ (if $\mathcal{T}$ makes a query $\xi \notin \tilde{S}$ then $\mathcal{V}_{\mathsf{amplify}}$ immediately rejects).
   (b) **Consistency Check:** For every $\xi \in \tilde{S}$, it holds that $\widetilde{\mathbf{chksum}}\big|_{\xi} = \mathsf{ENC}(\tilde{\phi}(\xi))$, where $\widetilde{\mathbf{chksum}} \stackrel{\text{def}}{=} \left(\widetilde{chksum}^{(1)}, \ldots, \widetilde{chksum}^{(\ell)}\right) \in \mathbb{F}^{(d \cdot \log(k)) \times (\ell \cdot a)}$ and $\tilde{\phi}(\xi) \in \mathbb{F}^k$ such that $(\tilde{\phi}(\xi))_j \stackrel{\text{def}}{=} \tilde{\phi}_j(\xi)$.

   If any test fails then $\mathcal{V}_{\mathsf{amplify}}$ immediately rejects.

5. $\mathcal{V}_{\mathsf{amplify}}$ outputs $w' = \left(\left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right), \tilde{S}\right)$ and the sequence $(x'_j)_{j \in [k]}$, where $x'_j \stackrel{\text{def}}{=} \tilde{\phi}_j$.

[a] Recall that the verifier has implicit access to the *low degree extension* of the main input (see Section 4.4).

Figure 1: The Deviation Amplification Protocol

Unambiguity of $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$. Let $w \in \{0,1\}^{n_{\mathsf{explicit}}}$, $x_1, \ldots, x_k \in \{0,1\}^n$, let $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ be a $(g, \mathbb{H}, \mathbb{F})$-encoded cheating prover strategy, let $i^* \in [\ell+1]$ and fix verifier messages $\beta^{(1)}, \ldots, \beta^{(i^*-1)} \in \mathbb{F}^b$ (corresponding to rounds $1, \ldots, i^*-1$) such that $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ on input $(w, (x_1, \ldots, x_k))$ and verifier messages $\beta^{(1)}, \ldots, \beta^{(i^*-1)}$ first deviates from the protocol at round i^*.

Recall that the matrices $A^{(1)}, \ldots, A^{(\ell)} \in \mathbb{F}^{k \times a}$, which would have been computed by the *prescribed* prover $\mathcal{P}_{\mathsf{amplify}}$ are defined as the matrices whose j^{th} row is $\alpha_j^{(i)} = \mathcal{P}_{\mathcal{L}}\left(w, x_j, \left(\beta^{(1)}, \ldots, \beta^{(i-1)}\right)\right)$. A crucial fact that will be used in our analysis is that these matrices are defined for arbitrary inputs $(x_1, \ldots, x_k)$ (i.e., even for inputs $(x_1, \ldots, x_k)$ for which $\exists j \in [k]$ s.t. $x_j \notin \mathcal{L}$).

The verifier $\mathcal{V}_{\mathsf{amplify}}$'s output at the end of the protocol (in case it does not reject) is $w' = \left((\beta^{(1)}, \ldots, \beta^{(\ell)}), \tilde{S}\right)$ and the sequence $(\tilde{\phi}_j)_{j \in [k]}$, where $\tilde{S} \subseteq [\ell \cdot a]$ and $\tilde{\phi}_j : \tilde{S} \to \mathbb{F}$. We say that $\mathcal{V}_{\mathsf{amplify}}$ wins if it either rejects or if there exists $\xi \in \tilde{S}$ such that $\Delta\left(\tilde{\phi}(\xi), \mathbf{A}|_\xi\right) \geq d/2$, where $\mathbf{A} = \left(A^{(1)}, \ldots, A^{(\ell)}\right) \in \mathbb{F}^{k \times (\ell \cdot a)}$ and $\tilde{\phi}(\xi) = \left(\tilde{\phi}_1(\xi), \ldots, \tilde{\phi}_k(\xi)\right)$.

If $\mathcal{V}_{\mathsf{amplify}}$ wins (and it does not reject), then there exists a set $J \subseteq [k]$ of size at least $|J| \geq d/2$ such that $\tilde{\phi}(\xi)$ and $\mathbf{A}|_\xi$ differ on their j^{th} coordinate, for every $j \in J$. Hence, by definition of $\mathcal{L}'$, for every $j \in J$ it holds that $\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right), \tilde{S}\right), (x_j, \phi_j)\right) \notin \mathcal{L}'$. Therefore, to prove Lemma 6.3 it suffices to show that $\mathcal{V}_{\mathsf{amplify}}$ wins with probability at least $\left(1 - \varepsilon - 2^{-\lambda}\right)$.

We first consider the case that $i^* \in [\ell]$ (i.e., $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ first deviates in Phase I) and later consider the simpler case that $i^* = \ell + 1$ (i.e., $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ first deviates in Phase II).

Case I: First Deviation in Round $i^* \in [\ell]$. For every $i \in [\ell]$, we denote by

$$chksum^{(i)} = \mathcal{P}_{\mathsf{amplify}}\left(w, (x_1, \ldots, x_k), i, \left(\beta^{(1)}, \ldots, \beta^{(i-1)}\right)\right) \in \mathbb{F}^{(d \cdot \log(k)) \times a}$$

and

$$\widetilde{chksum}^{(i)} = \widetilde{\mathcal{P}}_{\mathsf{amplify}}\left(w, (x_1, \ldots, x_k), i, \left(\beta^{(1)}, \ldots, \beta^{(i-1)}\right)\right) \in \mathbb{F}^{(d \cdot \log(k)) \times a}$$

the i^{th} message sent by the prescribed and cheating provers $\mathcal{P}_{\mathsf{amplify}}$ and $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ at round i, respectively. Note that these messages are fixed since we fixed $(\beta^{(1)}, \ldots, \beta^{(i-1)}$ and the prover is wlog deterministic. The fact that $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ deviates at round $i^* \in [\ell]$ means that $\widetilde{chksum}^{(i)} = chksum^{(i)}$ for every $i < i^*$, but $\widetilde{chksum}^{(i^*)} \neq chksum^{(i^*)}$. Note that since $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ is a $(g, \mathbb{H}, \mathbb{F})$-encoded prover strategy, each of the rows of each of the $\widetilde{chksum}^{(i)}$'s sent by $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ are composed of g vectors in $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ (recall that $m = \log_{|\mathbb{H}|}(a/g)$).

Using the matrices $A^{(1)}, \ldots, A^{(\ell)}$ (which are the matrices generated by the *prescribed* prover), we define corresponding matrices $\tilde{A}^{(1)}, \ldots, \tilde{A}^{(\ell)}$ as follows. For every $i \in [\ell]$, let $\tilde{A}^{(i)} \in \mathbb{F}^{k \times a}$ be defined as

$$\tilde{A}^{(i)} \stackrel{\text{def}}{=} \mathsf{DEC}\left(A^{(i)}, \widetilde{chksum}^{(i)}\right).$$

That is the columns of $\tilde{A}^{(i)}$'s are the closest to those of $A^{(i)}$'s that match the checksum $\widetilde{chksum}^{(i)}$. Note that for every $i < i^*$, it holds that $\tilde{A}^{(i)} = A^{(i)}$. For every $i \in [\ell]$ and $j \in [k]$, we denote the j^{th} row of $\tilde{A}^{(i)}$ by $\tilde{\alpha}_j^{(i)}$ (in analogy to $\alpha_j^{(i)}$ which is the j^{th} row of $A^{(i)}$).

Loosely speaking, our analysis will show that once the cheating prover $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ sends $\widetilde{chksum}^{(i)}$ it is "committed" to the corresponding matrix $\tilde{A}^{(i)}$, whose rows $\{\tilde{\alpha}_j^{(i)}\}_{j \in [k]}$ form messages for the i^{th} round of the underlying base protocols for $x_1, \ldots, x_k$.

A difficulty that we encounter here is that the $\tilde{\alpha}_j^{(i)}$ are not necessarily formed of g codewords of $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$. This is problematic, because we want to reduce to the unambiguity *against encoded provers* of the base protocol, by using the $\tilde{\alpha}_j^{(i)}$'s as our cheating prover's messages. We will overcome this difficulty by using the robustness of the transcript tester, see below.

We proceed to define an event E that will be central to our analysis.

Definition 6.7 (The Event E)**.** *Let E be the event that there exists $j^* \in [k]$ such that $\left(\tilde{\alpha}_{j^*}^{(1)}, \ldots, \tilde{\alpha}_{j^*}^{(\ell)}\right)$ is $\frac{1}{2\ell k g}$-far from the set:*

$$\left\{\left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right) \in ((\mathsf{LDE}_{\mathbb{F},\mathbb{H}_m})^g)^\ell \text{ such that } \mathcal{V}_{\mathcal{L}}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(\hat{x}_{j^*}, \left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right)\right)\right) = 1\right\},$$

where the probability is over $\beta^{(i^*)}, \ldots, \beta^{(\ell)}$, and $\hat{x}_j \stackrel{\text{def}}{=} \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_j)$.

That is, the event E, implies that for some $j^* \in [k]$ it holds that $\left(\tilde{\alpha}_{j^*}^{(1)}, \ldots, \tilde{\alpha}_{j^*}^{(\ell)}\right)$ is far from a sequence of valid messages (i.e., messages composed of a sequence of g codewords each) that form an accepting transcript for verifier $\mathcal{V}_L$ w.r.t. the input x_{j^*}.

We establish the unambiguity requirement for Case I using Proposition 6.8 and Proposition 6.9 below.

Proposition 6.8. $\Pr\left[E\right] \geq 1 - \varepsilon$.

Proof. Since $chksum^{(i^*)} \neq \widetilde{chksum}^{(i^*)}$, there must exist a row on which they differ. Denote these rows by τ and $\tilde{\tau}$, respectively. Since these two rows are each composed of g $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords (for τ by construction of $\mathcal{P}_{\mathsf{amplify}}$ and for $\tilde{\tau}$ by our assumption that $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ is (a, g)-encoded) and $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ has relative distance $1 - \frac{m(|\mathbb{H}|-1)}{|\mathbb{F}|} \geq \frac{1}{2}$ (by the Schwartz-Zippel Lemma, see Lemma 3.1), the rows τ and τ' must have relative distance at least $\frac{1}{2g}$.

Since ENC is a *linear* function, and $chksum^{(i^*)} = \mathsf{ENC}(A^{(i^*)})$ and $\widetilde{chksum}^{(i^*)} = \mathsf{ENC}(\tilde{A}^{(i^*)})$, it holds that τ and $\tilde{\tau}$ are linear combinations of the k rows of $A^{(i)}$ and $\tilde{A}^{(i)}$, respectively, with respect to the same coefficients (which are determined by the linear function ENC). Since τ and τ' differ on $\frac{1}{2g}$ fraction of their coordinates, there must exist a *fixed* row $j^* \in [k]$ on which $A^{(i^*)}$ and $\tilde{A}^{(i^*)}$ have relative distance at least $\frac{1}{2kg}$. In other words, $\alpha_{j^*}^{(i^*)}$ and $\tilde{\alpha}_{j^*}^{(i^*)}$ (which are, respectively, the $j^{*\text{th}}$ rows of $A^{(i^*)}$ and $\tilde{A}^{(i^*)}$) have relative distance at least $\frac{1}{2kg}$.

Proposition 6.8 follows from the following claim:

Claim 6.8.1. *The probability that $\left(\tilde{\alpha}_{j^*}^{(1)}, \ldots, \tilde{\alpha}_{j^*}^{(\ell)}\right)$ is $\frac{1}{2\ell k g}$ close to the set*

$$\left\{\left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right) \in ((\mathsf{LDE}_{\mathbb{F},\mathbb{H},m})^g)^\ell \;:\; \mathcal{V}_{\mathcal{L}}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(\hat{x}_{j^*}, \left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right)\right)\right) = 1\right\}$$

is less than ε, where the probability is over $\beta^{(i^*)}, \ldots, \beta^{(\ell)}$,

Proof. We prove Claim 6.8.1 by a reduction to the *unambiguity* of $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$.

Recall that $\beta^{(1)}, \ldots, \beta^{(i^*-1)}$ were already fixed. Let $\mathcal{B}$ be the set of all sequences $\left(\beta^{(i^*)}, \ldots, \beta^{(\ell)}\right) \in (\mathbb{F}^b)^{\ell - i^* + 1}$ such that there exists a sequence $\left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right) \in ((\mathsf{LDE}_{\mathbb{F},\mathbb{H},m})^g)^\ell$, that is $\frac{1}{2\ell k g}$ close to $\left(\tilde{\alpha}_{j^*}^{(1)}, \ldots, \tilde{\alpha}_{j^*}^{(\ell)}\right)$ and

$$\mathcal{V}_{\mathcal{L}}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(\hat{x}_{j^*}, \left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right)\right)\right) = 1.$$

To prove the claim we need to show that the density of $\mathcal{B}$ (within $(\mathbb{F}^b)^{\ell-i^*+1}$) is less than ε.

Consider the following cheating strategy $\widetilde{\mathcal{P}}_{\mathcal{L}}$ for the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$. For every $i \in [\ell]$, given as input $\left((w, x_{j^*}), i, \left(\beta^{(1)}, \ldots, \beta^{(i-1)}\right)\right)$, the cheating prover $\widetilde{\mathcal{P}}_{\mathcal{L}}$ uses $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ to compute $\tilde{\alpha}^{(i)}_{j^*}$ (recall that $\tilde{\alpha}^{(i)}_{j^*}$ is the $j^{*\text{th}}$ row of $\tilde{A}^{(i)}$) and finds a string $\pi^{(i)} \in \mathbb{F}^a$ such that:

- $\pi^{(i)}$ is $\frac{1}{2kg}$-close to $\tilde{\alpha}^{(i)}_{j^*}$; and
- $\pi^{(i)}$ is composed of g $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords (as noted above, it is not necessarily the case that the $\tilde{\alpha}^{(i)}_j$'s themselves are composed of g codewords).

(since $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ has relative distance $\left(1 - \frac{m(|\mathbb{H}|-1)}{|\mathbb{F}|}\right) \geq \frac{1}{2}$ there is at most one such sequence $\pi^{(i)}$). If no such string $\pi^{(i)}$ exists then $\widetilde{\mathcal{P}}_{\mathcal{L}}$ aborts. Otherwise, $\widetilde{\mathcal{P}}_{\mathcal{L}}$'s message at round i is $\pi^{(i)}$.

We show that if $\mathcal{V}_{\mathcal{L}}$ (w.r.t. implicit input x_{j^*}) is given the fixed messages[19] $\beta^{(1)}, \ldots, \beta^{(i^*-1)}$ for rounds $1, \ldots, i^* - 1$, respectively, it holds that:

- $\widetilde{\mathcal{P}}_{\mathcal{L}}$ first deviates from the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ at round i^*.
- $\mathcal{V}_{\mathcal{L}}$ accepts whenever it samples $\left(\beta^{(i^*)}, \ldots, \beta^{(\ell)}\right) \in \mathcal{B}$.

Hence, by the unambiguity of $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ we obtain that $\mathcal{B}$ has density less than ε.

To see that $\widetilde{\mathcal{P}}_{\mathcal{L}}$ does not deviate before round i^*, recall that $\tilde{\alpha}^{(i)}_{j^*} = \alpha^{(i)}_{j^*}$ for every $i < i^*$ (since $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ first deviates at round i^*). Since $\alpha^{(i)}_{j^*}$ are composed of g LDE codewords, it holds that $\pi^{(i)} = \alpha^{(i)}_{j^*}$ and so no deviation occurs at rounds $1, \ldots, i^* - 1$.

As for round i^*, we already showed that $\alpha^{(i^*)}_{j^*}$ and $\tilde{\alpha}^{(i^*)}_{j^*}$ are $\frac{1}{2kg}$-far. Since $\pi^{(i^*)}$ and $\tilde{\alpha}^{(i^*)}_{j^*}$ are $\frac{1}{2kg}$-close, we obtain that $\pi^{(i^*)} \neq \alpha^{(i^*)}_{j^*}$ and so $\widetilde{\mathcal{P}}_{\mathcal{L}}$ deviates at round i^*.

Lastly, we need to show that $\mathcal{V}_{\mathcal{L}}$ accepts whenever it samples $\left(\beta^{(i^*)}, \ldots, \beta^{(\ell)}\right) \in \mathcal{B}$, that is, $\left(\beta^{(i^*)}, \ldots, \beta^{(\ell)}\right)$ such that the $\tilde{\alpha}^{(i)}_j$'s are close to valid messages $\bar{\alpha}^{(i)}_j$'s (i.e., composed of g codewords each) that make the verifier accept.

Fix $\left(\beta^{(i^*)}, \ldots, \beta^{(\ell)}\right) \in \mathcal{B}$ and let $\left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right) \in ((\mathsf{LDE}_{\mathbb{F},\mathbb{H},m})^g)^\ell$, that are $\frac{1}{2\ell kg}$-close to $\left(\tilde{\alpha}^{(1)}_{j^*}, \ldots, \tilde{\alpha}^{(\ell)}_{j^*}\right)$ and

$$\mathcal{V}_{\mathcal{L}}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(\hat{x}_{j^*}, \left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right)\right)\right) = 1.$$

For every $i \in [\ell]$, $\bar{\alpha}^{(i)}$ is $\frac{1}{2kg}$-close to $\tilde{\alpha}^{(i)}_{j^*}$ (or otherwise the overall distance would be greater than $\frac{1}{2kg\ell}$). Furthermore, $\bar{\alpha}^{(i)}$ is composed of g LDE codewords. Hence, $\forall i \in [\ell]$, $\pi^{(i)} = \bar{\alpha}^{(i)}$ and so

$$\mathcal{V}_{\mathcal{L}}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(x_{j^*}, \left(\pi^{(1)}, \ldots, \pi^{(\ell)}\right)\right)\right) = 1.$$

and so, by the unambiguity of $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ we obtain that the density of $\mathcal{B}$ is less than ε. □

This concludes the proof of Proposition 6.8. □

[19] Recall that values $\beta^{(1)}, \ldots, \beta^{(i^*-1)}$, which are the messages that $\mathcal{V}_{\mathsf{amplify}}$ sends before $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ deviates, were already fixed.

Proposition 6.9.

$$\Pr\left[\mathcal{V}_{\mathsf{amplify}} \textit{ wins} \mid E\right] \geq 1 - 2^{-\lambda}$$

Proof. The fact that E occurs means that there exists some $j^* \in [k]$ such that $\left(\tilde{\alpha}_{j^*}^{(1)}, \ldots, \tilde{\alpha}_{j^*}^{(\ell)}\right)$ is $\frac{1}{2\ell k g}$-far from any $\left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right) \in ((\mathsf{LDE}_{\mathbb{F},\mathbb{H},m})^g)^\ell$ for which

$$\mathcal{V}_{\mathcal{L}}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(\hat{x}_j, \left(\bar{\alpha}^{(1)}, \ldots, \bar{\alpha}^{(\ell)}\right)\right)\right) = 1.$$

Hence, by the soundness condition in Lemma 6.5, with probability $1 - 2^{-\lambda}$, over the coins of the transcript tester $\mathcal{T}$ it holds that

$$\mathcal{T}\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right)\right), \left(\hat{x}_j, \left(\tilde{\alpha}_j^{(1)}, \ldots, \tilde{\alpha}_j^{(\ell)}\right)\right)\right) = 0. \tag{4}$$

Assume that the random string ρ chosen by $\mathcal{V}_{\mathsf{amplify}}$ in Step 2 is such that Eq. (4) holds (as noted above this happens with probability at least $1 - 2^{\lambda}$).

We assume that in Step 3, $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ sends a set $\tilde{S}$ and functions $\tilde{\phi}_1, \ldots, \tilde{\phi}_k : \tilde{S} \to \mathbb{F}$ so that all the queries that $\mathcal{T}$'s generates in Step 4a are answered (since otherwise $\mathcal{V}_{\mathsf{amplify}}$ rejects).

If in Step 3, $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ sends a function $\tilde{\phi}_j$ such that $\forall \xi \in \tilde{S}$, $\tilde{\phi}_j(\xi) = \tilde{\mathbf{a}}_j|_\xi$, where $\tilde{\mathbf{a}}_\mathbf{j} \stackrel{\text{def}}{=} \left(\tilde{\alpha}_j^{(1)}, \ldots, \tilde{\alpha}_j^{(\ell)}\right)$, then by Eq. (4) $\mathcal{V}_{\mathsf{amplify}}$ rejects in Step 4a and we are done. Hence, we assume that there exists $\xi \in \tilde{S}$ such that $\tilde{\mathbf{a}}_\mathbf{j}|_\xi \neq \tilde{\phi}_j(\xi)$, and in particular $\tilde{\mathbf{A}}|_\xi \neq \tilde{\phi}(\xi)$, where $\tilde{\phi}(\xi) \stackrel{\text{def}}{=} \left(\tilde{\phi}_1(\xi), \ldots, \tilde{\phi}_k(\xi)\right)$.

If $\mathsf{ENC}\left(\tilde{\phi}(\xi)\right) \neq \widetilde{\mathbf{chksum}}|_\xi$, where $\widetilde{\mathbf{chksum}} = \left(\widetilde{chksum}^{(1)}, \ldots, \widetilde{chksum}^{(\ell)}\right)$, then $\mathcal{V}_{\mathsf{amplify}}$ rejects in Step 4b and we are done, and so we assume that $\mathsf{ENC}\left(\tilde{\phi}(\xi)\right) = \widetilde{\mathbf{chksum}}|_\xi = \mathsf{ENC}\left(\tilde{\mathbf{A}}|_\xi\right)$.

Suppose for contradiction that $\Delta\left(\tilde{\phi}(\xi), \mathbf{A}|_\xi\right) \leq d/2$. Then,

$$\Delta\left(\tilde{\mathbf{A}}|_\xi, \mathbf{A}|_\xi\right) \leq \Delta\left(\tilde{\phi}(\xi), \mathbf{A}|_\xi\right) \leq d/2, \tag{5}$$

where $\tilde{\mathbf{A}} \stackrel{\text{def}}{=} \left(\tilde{A}^{(1)}, \ldots, \tilde{A}^{(\ell)}\right)$ and the first inequality follows from the fact that $\tilde{\mathbf{A}}|_\xi = \mathsf{DEC}\left(\mathbf{A}|_\xi, \widetilde{\mathbf{chksum}}|_\xi\right)$ is defined as the vector that is closest to $\mathbf{A}|_\xi$ with the same checksum value.

Hence, by the triangle inequality,

$$\Delta\left(\tilde{\phi}(\xi), \tilde{\mathbf{A}}|_\xi\right) \leq \Delta\left(\tilde{\phi}(\xi), \mathbf{A}|_\xi\right) + \Delta\left(\mathbf{A}|_\xi, \tilde{\mathbf{A}}|_\xi\right) \leq d. \tag{6}$$

and so $\tilde{\phi}(\xi)$ and $\tilde{\mathbf{A}}|_\xi$ are distinct vectors with the same checksum value, but $\Delta\left(\tilde{\phi}(\xi), \tilde{\mathbf{A}}|_\xi\right) \leq d$ in contradiction to the fact that C has distance $d+1$. Hence, $\Delta\left(\tilde{\phi}(\xi), \mathbf{A}|_\xi\right) > d/2$ which means that $\mathcal{V}_{\mathsf{amplify}}$ wins.

This completes the proof of Proposition 6.9. □

By combining Proposition 6.8 and Proposition 6.9 we obtain that:

$$\Pr\left[\mathcal{V}_{\mathsf{amplify}} \text{ wins}\right] \geq \Pr[E] \cdot \Pr\left[\mathcal{V}_{\mathsf{amplify}} \text{ wins} \mid E\right] \geq 1 - \varepsilon - 2^{-\lambda}.$$

This completes the analysis of Case I.

Case II: First Deviation in Round $i^* = \ell + 1$. Since $\widetilde{\mathcal{P}}_{\mathsf{amplify}}$ deviates at round $\ell+1$, either it deviates on the padding (in which case $\mathcal{V}_{\mathsf{amplify}}$ rejects with probability $1 - 2^{-\lambda}$, see Remark 4.13) or there exists $j \in [k]$ and $\xi \in \tilde{S}$ such that $\tilde{\phi}_j(\xi) \neq \phi_j(\xi)$ and in particular, $\tilde{\phi}(\xi) \neq \phi(\xi) = \mathbf{A}_\xi$ (a third option is that $\tilde{S} \neq S$ but $\forall j \in [k],\ \tilde{\phi}_j \equiv \phi_j$ but in this case $\mathcal{V}_{\mathsf{amplify}}$ rejects in 4a when $\mathcal{T}$ queries a point not in $\tilde{S}$).

If $\mathsf{ENC}\left(\tilde{\phi}(\xi)\right) \neq \mathbf{chksum}|_\xi = \mathsf{ENC}\left(\mathbf{A}|_\xi\right)$, then $\mathcal{V}_{\mathsf{amplify}}$ rejects in Step 4b and we are done. Otherwise, $\mathsf{ENC}\left(\tilde{\phi}(\xi)\right) = \mathsf{ENC}\left(\mathbf{A}|_\xi\right)$ and since the code C has distance $d+1$ (and the two messages are distinct), it must be the case that $\Delta\left(\tilde{\phi}(\xi), \mathbf{A}|_\xi\right) \geq d+1$ and in particular $\mathcal{V}_{\mathsf{amplify}}$ wins. This completes the analysis of Case II.

This completes the analysis of the unambiguity of $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$.

Complexity Measures of $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$. By construction, in Phase I $\mathcal{V}_{\mathsf{amplify}}$ sends messages of length b and in Phase II (in Step 2) it sends a single message which consists of random coins for the transcript tester. By Lemma 6.5 the tester uses $k \cdot \mathsf{poly}(|\mathbb{H}|, g, \ell, \lambda)$ random coins and so each verifier message is of length $\max\left(b, k \cdot \mathsf{poly}(|\mathbb{H}|, g, \ell, \lambda)\right)$.

In Phase I (Step 3) $\mathcal{P}_{\mathsf{amplify}}$ sends messages of length $d \cdot \log(k) \cdot a$ (these messages are of the correct format since they are linear combinations of such messages) and in Phase II it sends a message of length $k \cdot |S|$. By our assumption $\mathsf{poly}(k \cdot |S|) \leq a$ and so each message sent is of length at most $d \cdot \log(k) \cdot a$.

In Phase I $\mathcal{V}_{\mathsf{amplify}}$ makes no queries (neither to the input nor to the transcript) and no computation (recall that in the PCIP model receiving messages does not count toward running time). In Phase II, in Step 4a, $\mathcal{V}_{\mathsf{amplify}}$ runs the transcript tester k times, for a total runtime of

$$k \cdot \left(\mathcal{V}\mathsf{time} + q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\right)$$

while making $k \cdot \left(q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\right)$ queries to the transcript and to the implicit input.

In Step 4b $\mathcal{V}_{\mathsf{amplify}}$ reads the set $\tilde{S}$, the entire (truth tables of the) functions $\tilde{\phi}_1, \ldots, \tilde{\phi}_k$ and the relevant columns of $\widetilde{chksum}$. This amounts to $O(k) \cdot \left(q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\right)$ queries (and runtime). Furthermore, the verifier computes the corresponding checksum values and this takes additional time

$$k \cdot \mathsf{poly}(\log(|\mathbb{F}|), \log(k)) \cdot \left(q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\right) = k \cdot \left(q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\right).$$

Checking the padding adds an additional $O(\ell \cdot g \cdot \mathsf{poly}(|\mathbb{H}|) \cdot \lambda)$ to the runtime and queries. Thus, overall $\mathcal{V}_{\mathsf{amplify}}$ runs in time:

$$k \cdot \mathcal{V}\mathsf{time} + k \cdot q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k^2 \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)$$

and makes at most

$$k \cdot q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k^2 \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)$$

queries to the transcript and implicit input.

As for $\mathcal{P}_{\mathsf{amplify}}$'s runtime, in Phase I it runs $\mathcal{P}_{\mathcal{L}}$ k times and computes the checksum values for a total runtime of $k \cdot \mathcal{P}\mathsf{time} + \ell \cdot a \cdot k \cdot \mathsf{polylog}(|\mathbb{F}|) \leq 2k \cdot \mathcal{P}\mathsf{time}$ (by our assumption that

$\mathcal{P}\mathsf{time} \geq \ell \cdot a \cdot \mathsf{polylog}(|\mathbb{F}|)$). In Phase II (3), $\mathcal{P}_{\mathsf{amplify}}$ needs to generate $\mathcal{T}$'s queries, and then to send the values of the original messages $\alpha^{(i)})_j$ restricted to the foregoing query coordinates. The latter can be done in $k \cdot (\mathcal{V}\mathsf{time} + q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|))$ time. Thus, overall $\mathcal{P}_{\mathsf{amplify}}$ runs in time $\big(2k \cdot \mathcal{P}\mathsf{time} + k \cdot \mathcal{V}\mathsf{time} + k \cdot q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k^2 \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)\big)$.

6.2.3 PCIP for the Language $\mathcal{L}'$

So far we have established Condition 2 in the statement of Lemma 6.3. In order to complete the proof of Lemma 6.3 we still need to establish Condition 1, that is, to show that $\mathcal{L}'$ has an unambiguous PCIP w.r.t. encoded provers (with the appropriate parameters).

Recall that our goal is to construct an unambiguous PCIP $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$ for the language $\mathcal{L}'$. Thus, we need to be able to check that a given input x belongs to $\mathcal{L}$ *and* that the messages sent by the prescribed prover $\mathcal{P}_{\mathcal{L}}$ with respect to a fixed sequence of verifier random coins $\beta^{(1)}, \ldots, \beta^{(\ell)}$, projected to a particular set of coordinates S, is equal to a specific set of values (see the definition of $\mathcal{L}'$ above for details).

A naive (and unsound) approach for such a protocol is to have the prover $\mathcal{P}_{\mathcal{L}'}$ send the messages that $\mathcal{P}_{\mathcal{L}}$ would have sent in the interaction and to check that these messages (1) make $V_{\mathcal{L}}$ accept, and (2) that their projection to S is equal to the desired values. The problem is that the coins for $\mathcal{V}_{\mathcal{L}}$ in this interaction were already fixed (as part of the input to $\mathcal{L}'$. Interactive proofs in general, and here specifically, do not provide any soundness (much less unambiguity) against cheating provers who know future coin tosses.

Thus, we seek a way to check that messages sent by the prover above are indeed the messages that $\mathcal{P}_{\mathcal{L}}$ would have sent in the interaction. The key observation is that *unambiguity* gives us a method to check that a particular message sent by the prover at a particular round is in fact the prescribed message - just continue the $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ interaction from that round *using fresh random coins*. If the message sent is not the prescribed message then the unambiguity of the base PCIP guarantees that the verifier will reject.

Thus, for every round $i \in [\ell]$ of the original protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ we will run a random continuation of that protocol, where until round i we use the fixed strings from the description of $\mathcal{L}'$ (i.e., $\beta^{(1)}, \ldots, \beta^{(i-1)}$), and from that point we use fresh random strings (i.e., using fresh random coins $\gamma^{(i)}, \ldots, \gamma^{(\ell)}$). We are, essentially, running ℓ parallel copies of the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ using different random coins for each copy. (Since we do not need these copies to be independent, we allow ourselves to use the same fresh random coins $\gamma^{(1)}, \ldots, \gamma^{(\ell)}$ for all of the ℓ protocols.)

Before presenting the protocol, a few words about the notation that we use. In the protocol, we denote the prover's messages by $\alpha^{(i_1, i_2)}$ for $i_2 \leq i_1$. This notation refers to messages generated by the prescribed prover $\mathcal{P}_{\mathcal{L}}$ in round i_1 in the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$, when the messages sent by the verifier are $\beta^{(1)}, \ldots, \beta^{(i_2-1)}, \gamma^{(i_2)}, \ldots, \gamma^{(i_1-1)}$. That is, the first $i_2 - 1$ verifier messages are those specified as part of the explicit input to $\mathcal{L}'$ and the latter $i_1 - i_2$ messages consist of fresh random coins (which are generated within the protocol for $\mathcal{L}'$). The unambiguous PCIP for $\mathcal{L}'$ is presented in Fig. 2.

Prescribed Completeness for $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$. All the tests that $\mathcal{V}_{\mathcal{L}'}$ runs in Step 2 of the protocol correspond to executions of the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ with respect to the input (w, x), where, for every $i_2 \in [\ell]$, the $i_2{}^{\text{th}}$ test corresponds to the sequence $\big(\beta^{(1)}, \ldots, \beta^{(i_2-1)}, \gamma^{(i_2)}, \ldots, \gamma^{(\ell)}\big)$ of messages "sent by" $\mathcal{V}_{\mathcal{L}}$. By the prescribed completeness of $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ these tests pass if and only if $(w, x) \in \mathcal{L}$.

Unambiguous Interactive Proof $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$ for $\mathcal{L}'$

Prover's Input: $x \in \{0,1\}^n$, $w \in \{0,1\}^{n_{\text{explicit}}}$, $\beta^{(1)}, \ldots, \beta^{(\ell)} \in \mathbb{F}^b$, $S \subseteq [\ell \cdot a]$ and $\phi : S \to \mathbb{F}$.
Verifier's Input: explicit access to $\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right), S\right)$ and implicit access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x, \phi)$.

1. For $i_1 = 1, \ldots, \ell$:
   (at the beginning of round i_1 the prover $\mathcal{P}_{\mathcal{L}'}$ already has $\gamma^{(1)}, \ldots, \gamma^{(i_1-1)} \in \mathbb{F}^b$)
   (a) For $i_2 = 1, \ldots, i_1$: the prover $\mathcal{P}_{\mathcal{L}'}$ computes the message $\alpha^{(i_1,i_2)} = \mathcal{P}_{\mathcal{L}}\left((x,w), i_1, \left(\beta^{(1)}, \ldots, \beta^{(i_2-1)}, \gamma^{(i_2)}, \ldots, \gamma^{(i_1-1)}\right)\right) \in \mathbb{F}^a$.
   (b) $\mathcal{P}_{\mathcal{L}'}$ sends the message $\left(\alpha^{(i_1,1)}, \ldots, \alpha^{(i_1,i_1)}\right)$ to $\mathcal{V}_{\mathcal{L}'}$, using suitable padding (see Remark 4.13).
   (c) $\mathcal{V}_{\mathcal{L}'}$ receives the message $\left(\tilde{\alpha}^{(i_1,1)}, \ldots, \tilde{\alpha}^{(i_1,i_1)}\right)$ (which is allegedly equal to $\left(\alpha^{(i_1,1)}, \ldots, \alpha^{(i_1,i_1)}\right)$) and checks the padding (see Remark 4.13)
   (d) $\mathcal{V}_{\mathcal{L}'}$ chooses uniformly at random a string $\gamma^{(i_1)} \in \mathbb{F}^b$ and sends $\gamma^{(i_1)}$ to $\mathcal{P}_{\mathcal{L}'}$.
2. For $i_2 = 1, \ldots, \ell$:
   $\mathcal{V}_{\mathcal{L}'}$ runs $\mathcal{V}_{\mathcal{L}}$ on explicit input w, using the random strings $\left(\beta^{(1)}, \ldots, \beta^{(i_2-1)}, \gamma^{(i_2)}, \ldots, \gamma^{(\ell)}\right)$ while answering $\mathcal{V}_{\mathcal{L}}$'s queries to the transcript queries using $\left(\tilde{\alpha}^{(1,1)}, \ldots, \tilde{\alpha}^{(i_2,i_2)}, \tilde{\alpha}^{(i_2+1,i_2)}, \ldots, \tilde{\alpha}^{(\ell,i_2)}\right)$. Input queries to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$ are answered using the procedure in Proposition 3.8 applied to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x, \phi)$. If any of the tests fail then $\mathcal{V}_{\mathcal{L}'}$ immediately rejects.
3. $\mathcal{V}_{\mathcal{L}'}$ checks that for every $\xi \in S$ it holds that $\phi(\xi) = \left(\tilde{\alpha}^{(1,1)}, \ldots, \tilde{\alpha}^{(\ell,\ell)}\right)\big|_{\xi}$. If any of the tests fail then $\mathcal{V}_{\mathcal{L}'}$ rejects, otherwise it accepts (note that $\mathcal{V}_{\mathcal{L}'}$ uses $|\phi| = |S| \cdot \log(|\mathbb{F}|)$ queries to read all of ϕ from $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x, \phi)$).

Figure 2: Interactive Proof for $\mathcal{L}'$

For Step 3, since for every $i_1 \in [\ell]$ it holds that $\tilde{\alpha}^{(i_1,i_1)} = \alpha^{(i_1,i_1)} = \mathcal{P}_{\mathcal{L}}\left(x, \left(\beta^{(1)}, \ldots, \beta^{(i_1-1)}\right)\right)$, the tests in Step 3 pass if and only if for every $i_1 \in [\ell]$ and $\xi \in S$ it holds that $\left(\alpha^{(1)}, \ldots, \alpha^{(\ell)}\right)\big|_{\xi} = \phi^{(i_1)}(\xi)$, where $\alpha^{(i_1)} = \mathcal{P}_{\mathcal{L}}\left((x,w), i_1, \left(\beta^{(1)}, \ldots, \beta^{(i_1-1)}\right)\right)$.

We conclude that $\mathcal{V}_{\mathcal{L}'}$ accepts if and only if $x \in \mathcal{L}$ and $\left(\alpha^{(1)}, \ldots, \alpha^{(\ell)}\right)\big|_{\xi} = \phi(\xi)$, for every $\xi \in S$. That is, $\mathcal{V}_{\mathcal{L}'}$ accepts if and only if $\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right), S\right), (x, \phi)\right) \in \mathcal{L}'$.

The queries that $\mathcal{V}_{\mathcal{L}}$ makes are ℓ executions of $\mathcal{V}_{\mathcal{L}}$ and in addition reading the points in S from the transcript. The latter are input-oblivious since S is part of the explicit input.

Unambiguity for $(\mathcal{P}_{\mathcal{L}'}, \mathcal{V}_{\mathcal{L}'})$. Let $x \in \{0,1\}^n$, $w \in \{0,1\}^{n_{\mathsf{explicit}}}$, $\beta^{(1)}, \ldots, \beta^{(\ell)} \in \mathbb{F}^b$, $S \subseteq [\ell \cdot a]$ and $\phi : S \to \mathbb{F}$. Let $\widetilde{\mathcal{P}}_{\mathcal{L}'}$ be an $(\ell \cdot g, \mathbb{H}, \mathbb{F})$-encoded cheating prover strategy, $i_1^* \in [\ell]$ be a round, $\gamma^{(1)}, \ldots, \gamma^{(i_1^*-1)} \in \mathbb{F}^b$ be messages for $\mathcal{V}_{\mathcal{L}'}$ for rounds $1, \ldots, i_1^* - 1$ such that $\widetilde{\mathcal{P}}_{\mathcal{L}'}$ first deviates from $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ at round i_1^*, on input $\left(\left(w, \left(\beta^{(1)}, \ldots, \beta^{(\ell)}\right), S\right), (x, \phi)\right)$ and verifier messages $\gamma^{(1)}, \ldots, \gamma^{(i_1^*-1)}$. We stress that the messages $\beta^{(1)}, \ldots, \beta^{(\ell)}$ are fixed messages that are part of the (explicit) input, the messages $\gamma^{(1)}, \ldots, \gamma^{(i_1*-1)}$ are also fixed messages for $\mathcal{V}_{\mathcal{L}'}$ in rounds $1, \ldots, i^*-1$ and that $\gamma^{(i_1^*)}, \ldots, \gamma^{(\ell)}$ are *random* messages, chosen by $\mathcal{V}_{\mathcal{L}'}$ in rounds $i_1^*, \ldots, \ell$.

If $\widetilde{\mathcal{P}}_{\mathcal{L}'}$ deviates on the padding, then by Remark 4.13 $\mathcal{V}_{\mathcal{L}'}$ rejects with probability $1 - 2^{-\lambda}$. Otherwise, $\exists i_2^* \in [i_1^*]$ such that $\tilde{\alpha}^{(i_1^*,i_2^*)} \neq \alpha^{(i_1^*,i_2^*)} \stackrel{\text{def}}{=} \mathcal{P}_{\mathcal{L}}\left((x,w), i_1^*, \left(\beta^{(1)}, \ldots, \beta^{(i_2^*-1)}, \gamma^{i_2^*}, \ldots, \gamma^{(i_1^*-1)}\right)\right)$. Suppose toward a contradiction that the probability, over $\gamma^{(i_1^*)}, \ldots, \gamma^{(\ell)}$ that $\mathcal{V}_{\mathcal{L}'}$ accepts is greater than ε.

We derive a contradiction by constructing a $(g, \mathbb{H}, \mathbb{F})$-encoded cheating prover strategy $\widetilde{\mathcal{P}}_{\mathcal{L}}$ for the protocol $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ with respect to input (x,w), round i_1^* and fixed messages $\beta^{(1)}, \ldots, \beta^{(i_2^*-1)}, \gamma^{(i_2^*)}, \ldots, \gamma^{(i_1^*-1)} \in \mathbb{F}^b$ for $\mathcal{V}_{\mathcal{L}}$, a contradiction to the unambiguity of $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$.

For every $i_1 \in [\ell]$, given as input $\left((x,w), i_1, \left(\beta^{(1)}, \ldots, \beta^{(i_2^*-1)}, \gamma^{(i_2^*)}, \ldots, \gamma^{(i_1-1)}\right)\right)$, the prover $\widetilde{\mathcal{P}}_{\mathcal{L}}$ runs $\widetilde{\mathcal{P}}_{\mathcal{L}'}\left((x,w), i_1, \left(\gamma^{(1)}, \ldots, \gamma^{(i_1-1)}\right)\right)$ to obtain the message $\left(\tilde{\alpha}^{(i_1,1)}, \ldots, \tilde{\alpha}^{(i_1,i_1)}\right)$. If $i_1 < i_2^*$, the prover $\widetilde{\mathcal{P}}_{\mathcal{L}}$ outputs the message $\bar{\alpha}^{(i_1)} \stackrel{\text{def}}{=} \tilde{\alpha}^{(i_1,i_1)}$, otherwise (i.e., $i_1 \in [i_2^*, \ell]$), it outputs $\bar{\alpha}^{(i_1)} \stackrel{\text{def}}{=} \tilde{\alpha}^{(i_1,i_2^*)}$. Note that in both cases, since $\widetilde{\mathcal{P}}_{\mathcal{L}'}$ is $(\ell \cdot g, \mathbb{H}, \mathbb{F})$-encoded, the message $\bar{\alpha}^{(i_1)}$ is in $(\mathsf{LDE}_{\mathbb{F},\mathbb{H},m})^g$ and so $\widetilde{\mathcal{P}}_{\mathcal{L}}$ is $(g, \mathbb{H}, \mathbb{F})$-encoded.

We show that, with respect to the fixed verifier messages $\beta^{(1)}, \ldots, \beta^{(i_2^*-1)}, \gamma^{(i_2^*)}, \ldots, \gamma^{(i_1^*-1)}$ for $\mathcal{V}_{\mathcal{L}}$ in rounds $1, \ldots, i_1^* - 1$, it holds that $\widetilde{\mathcal{P}}_{\mathcal{L}}$ (first) deviates at round i_1^* but $\mathcal{V}_{\mathcal{L}}$ accepts with probability greater than ε, where the probability is over $\mathcal{V}_{\mathcal{L}}$'s random coins $\gamma^{(i_1^*)}, \ldots, \gamma^{(\ell)}$ for rounds $i_1^*, \ldots, \ell$.

To see that $\widetilde{\mathcal{P}}_{\mathcal{L}}$ does not deviate at any round $i_1 < i_1^*$, observe that, since $\widetilde{\mathcal{P}}_{\mathcal{L}'}$ does not deviate before round i_1^*, the message of $\widetilde{\mathcal{P}}_{\mathcal{L}}$ at round $i_1 \in [i_2^* - 1]$ is $\bar{\alpha}^{(i_1)} = \tilde{\alpha}^{(i_1,i_1)} = \alpha^{(i_1,i_1)} = \mathcal{P}_{\mathcal{L}}\left((x,w), i_1, \left(\beta^{(1)}, \ldots, \beta^{(i_1-1)}\right)\right)$ and its message at round $i_1 \in [i_2^*, i_1^* - 1]$ is $\bar{\alpha}^{(i_1)} = \tilde{\alpha}^{(i_1,i_2^*)} = \alpha^{(i_1,i_2^*)} = \mathcal{P}_{\mathcal{L}}\left((x,w), i_1, \left(\beta^{(1)}, \ldots, \beta^{(i_2^*-1)}, \gamma^{(i_2^*)}, \ldots, \gamma^{(i_1-1)}\right)\right)$.

At round i_1^*, the message that $\mathcal{P}_{\mathcal{L}'}$ computes is

$$\bar{\alpha}^{(i_1^*)} = \tilde{\alpha}^{(i_1^*,i_2^*)} \neq \alpha^{(i_1^*,i_2^*)} = \mathcal{P}_{\mathcal{L}}\left((x,w), i_1, \left(\beta^{(1)}, \ldots, \beta^{(i_2^*-1)}, \gamma^{(i_2^*)}, \ldots, \gamma^{(i_1^*-1)}\right)\right),$$

which is indeed a deviation from $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$.

Observe that,

$$\begin{aligned}
\varepsilon &< \Pr[\mathcal{V}_{\mathcal{L}'} \text{ accepts}] \\
&\leq \Pr\left[\mathcal{V}_{\mathcal{L}}\left(\left(w\left(\beta^{(1)},\ldots,\beta^{(i_2^*-1)},\gamma^{(i_2^*)},\ldots,\gamma^{(\ell)}\right)\right),\left(\hat{x},\left(\tilde{\alpha}^{(1,1)},\ldots,\tilde{\alpha}^{(i_2^*,i_2^*)},\tilde{\alpha}^{(i_2^*+1,i_2^*)},\ldots,\tilde{\alpha}^{(\ell,i_2^*)}\right)\right)\right)=1\right] \\
&= \Pr\left[\mathcal{V}_{\mathcal{L}}\left(\left(w\left(\beta^{(1)},\ldots,\beta^{(i_2^*-1)},\gamma^{(i_2^*)},\ldots,\gamma^{(\ell)}\right)\right),\left(\hat{x},\left(\bar{\alpha}^{(1)},\ldots,\bar{\alpha}^{(\ell)}\right)\right)\right)=1\right].
\end{aligned}$$

where $\hat{x} = \mathsf{LDE}_{\mathbb{F},\mathbb{H},m}(x)$, the first inequality is by our assumption on $\widetilde{\mathcal{P}}_{\mathcal{L}'}$ and the second inequality follows from the fact that $\mathcal{V}_{\mathcal{L}'}$ explicitly checks this condition.

Thus, $\widetilde{\mathcal{P}}_{\mathcal{L}}$ breaks the unambiguity of $(\mathcal{P}_{\mathcal{L}}, \mathcal{V}_{\mathcal{L}})$ in contradiction to our assumption.

Complexity Measures for $\mathcal{L}'$. By construction the number of rounds is $\ell' = \ell$, $\mathcal{V}_{\mathcal{L}'}$ sends messages of length $b' = b$ and $\mathcal{P}_{\mathcal{L}'}$ sends messages which are composed of at most ℓ messages generated by $\mathcal{P}_{\mathcal{L}}$ and so each message is of length $a' = \ell \cdot a$ and has $g' = \ell \cdot g$ (using sufficient padding).

The verifier $\mathcal{V}_{\mathcal{L}'}$ runs $\mathcal{V}_{\mathcal{L}}$ ℓ times and in addition reads $|S|$ field elements from the transcript and $|S| \cdot \log(|\mathbb{F}|)$ bits from the input (using Proposition 3.8) and compares the two. In addition $\mathcal{V}_{\mathcal{L}'}$ checks the padding (see Remark 4.13) for an additional $\ell \cdot g \cdot \mathsf{poly}(|\mathbb{H}|) \cdot \lambda$ queries and runtime.

Thus, $\mathcal{V}_{\mathcal{L}'}$ makes a total of

$$q' = \ell \cdot q + O(|S| \cdot \log(|\mathbb{F}|)) + \ell \cdot g \cdot \mathsf{poly}(|\mathbb{H}|) \cdot \lambda = q \cdot \left(\ell + \mathsf{poly}(\lambda, |\mathbb{H}|)\right) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|)$$

queries and runs in time

$$\mathcal{V}\mathsf{time}' = \ell \cdot \mathcal{V}\mathsf{time} + O(|S| \cdot \log(|\mathbb{F}|)) + \ell \cdot g \cdot \mathsf{poly}(|\mathbb{H}|) \cdot \lambda = \ell \cdot \mathcal{V}\mathsf{time} + q \cdot \mathsf{poly}(\lambda, |\mathbb{H}|) + k \cdot \mathsf{poly}(\ell, g, \lambda, |\mathbb{H}|).$$

Lastly, the prover $\mathcal{P}_{\mathcal{L}'}$ just runs $\mathcal{P}_{\mathcal{L}}$ exactly ℓ times and so $\mathcal{P}\mathsf{time}' = \ell \cdot \mathcal{P}\mathsf{time}$.
This concludes the proof of Lemma 6.3.

6.3 Proof of the Batch Verification Lemma (Lemma 6.1)

Using Lemma 6.3 we are now ready to prove Lemma 6.1. Recall that our basic idea for batching k protocols (with respect to k inputs) is to iteratively apply the deviation amplification lemma. More precisely, we will maintain a set of "active" inputs and iteratively shrink this set. The iterative shrinking step is accomplished by applying the deviation amplification protocol to the set of currently active inputs, and then sub-sampling from this set for the next iteration. Since the deviation amplification protocol increases the number of "false" inputs, sub-sampling from the current set of active inputs allows us to still catch at least one "false" input with high probability. Once the set of active inputs is sufficiently small, we are left with relatively few active inputs, each of which has an unambiguous PCIPs, and we simply run these PCIPs directly.

The batch verification protocol offers a trade-off between the number of rounds and total communication. This trade-off is governed by a parameter μ, which corresponds to the number of iterations of the shrinking step above. In each iteration, we shrink the set of active inputs by (roughly) a $k^{1/\mu}$ multiplicative factor. Since we start with k inputs, after μ iterations the set of active inputs will be sufficiently small to run their PCIPs. The total round complexity of the batch verification protocol grows proportionally to μ, since each of the (sequential) iterations requires multiple rounds of interaction.

Throughout this subsection, we use the convention that superscripts $i \in [\mu]$ correspond to the aforementioned iterations and subscripts $j \in [k]$ correspond to the k original inputs $x_1, \ldots, x_k$.

In order to present the batch verification theorem we first need to specify the μ protocols that we obtain by invoking Lemma 6.3 μ times. Recall that $\mathcal{L}$ is the (pair) language for which we assumed the existence of an unambiguous PCIP and that our goal is to construct a (non-trivial) unambiguous PCIP for $\mathcal{L}^{\otimes k}$. Let $\mu \geq 1$ be a parameter (controlling the trade-off between rounds and communication), let λ be a security parameter and let $d \stackrel{\text{def}}{=} k^{1/\mu} \cdot 2\lambda$.

We denote $\mathcal{L}^{(0)} \stackrel{\text{def}}{=} \mathcal{L}$ and $k^{(0)} \stackrel{\text{def}}{=} k$. For every $i \in [\mu]$, consider the deviation amplification lemma (Lemma 6.3) with respect to the language $\mathcal{L}^{(i-1)}$, number of inputs $k^{(i-1)}$ and parameters $d \stackrel{\text{def}}{=} k^{1/\mu} \cdot 2\lambda$ and security parameter λ. Let $(\mathcal{L}^{(i-1)})'$ be the language guaranteed by the deviation amplification lemma. We denote $\mathcal{L}^{(i)} = (\mathcal{L}^{(i-1)})'$ and $k^{(i)} = \frac{2k^{(i-1)}}{d} \cdot \lambda = k^{1-(i/\mu)}$. The following lemma shows that each one of the languages $\mathcal{L}^{(0)}, \ldots, \mathcal{L}^{(\mu)}$ has an unambiguous PCIP.

Lemma 6.10. *For every $i \in \{0, \ldots, \mu\}$, the language $\mathcal{L}^{(i)}$ (defined above) has an $\varepsilon^{(i)}$-unambiguous $(q^{(i)}, \ell^{(i)}, b^{(i)}, a^{(i)}, \mathcal{P}\mathsf{time}^{(i)}, \mathcal{V}\mathsf{time}^{(i)})$-$\mathsf{PCIP}$ $(\mathcal{P}_{\mathcal{L}^{(i)}}, \mathcal{V}_{\mathcal{L}^{(i)}})$ w.r.t. $(g^{(i)}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and the following parameters:*

- $\varepsilon^{(i)} = \varepsilon + i \cdot 2^{-\lambda}$.
- $q^{(i)} \leq \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{i+1} \cdot \Big(q + k \cdot \mathsf{poly}(g)\Big)$
- $\ell^{(i)} = \ell$.
- $b^{(i)} = b$.
- $a^{(i)} = \ell^i \cdot a$.
- $\mathcal{P}\mathsf{time}^{(i)} = \ell^i \cdot \mathcal{P}\mathsf{time}$.
- $\mathcal{V}\mathsf{time}^{(i)} \leq \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{i+1} \cdot \Big(\mathcal{V}\mathsf{time} + q + k \cdot \mathsf{poly}(g)\Big)$.
- $g^{(i)} = \ell^i \cdot g$.

Proof. By induction on i and using Lemma 6.3. Note that to use Lemma 6.3 we have to guarantee that:

1. $\log(a^{(i)}) \leq \min\Big(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\Big)$.
2. $a^{(i)} \geq \mathsf{poly}(k^{(i)}, q^{(i)}, \ell^{(i)}, g^{(i)}, \lambda, |\mathbb{H}|)$.
3. $\mathcal{P}\mathsf{time}^{(i)} \geq \ell^{(i)} \cdot a^{(i)} \cdot \mathsf{polylog}(|\mathbb{F}|)$.

The latter follows from our assumption on the parameters (specifically our assumptions that (1) $\log(\ell^\mu \cdot a) \leq \min\Big(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\Big)$, (2) $a \geq \mathsf{poly}(k, q, g) \cdot (\mathsf{poly}(\lambda, |\mathbb{H}|, \ell))^\mu$, and (3) $\mathcal{P}\mathsf{time} \geq \ell \cdot a \cdot \mathsf{polylog}(|\mathbb{F}|)$). □

Using the above definitions and Lemma 6.10, the batch verification protocol $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$ is described in Fig. 3.

Batch Verification Protocol

Round parameter: $\mu \geq 0$. Security parameter: $\lambda \geq 1$.
Prover's Input: $w \in \{0,1\}^{n_{\text{explicit}}}$ and $x_1, \ldots, x_k \in \{0,1\}^n$.
Verifier's Input: explicit access to w and implicit access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$.

(The languages $\mathcal{L}^{(i)}$ and the corresponding protocols $(\mathcal{P}_{\mathcal{L}^{(i)}}, \mathcal{V}_{\mathcal{L}^{(i)}}$ that appear below are defined in the beginning of Section 6.3)

1. Set $S^{(0)} \stackrel{\text{def}}{=} [k]$, $w^{(0)} \stackrel{\text{def}}{=} w$ and $x_j^{(0)} \stackrel{\text{def}}{=} x_j$, for every $j \in [k]$. Also set $d \stackrel{\text{def}}{=} k^{1/\mu} \cdot 2\lambda$.
2. For $i = 0, \ldots, \mu - 1$:
   (We maintain the invariant that at the end of round i, both parties know $S^{(i+1)}$, an explicit input $w^{(i+1)}$ and have oracle access to $\mathsf{LDE}(x_j^{(i+1)})$, for every $j \in S^{(i+1)}$)
   (a) $\mathcal{P}_{\mathsf{Batch}}$ and $\mathcal{V}_{\mathsf{Batch}}$ run the deviation amplification protocol $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$ on input $\left(w^{(i)}, (x_j^{(i)})_{j \in S^{(i)}}\right)$, with respect to the language $\mathcal{L}^{(i)}$, the parameter d and security parameter λ.
   Input queries to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_j^{(i)})$ are simulated using the above invariant. The prover's messages in the protocol are padded to length $a^{(\mu)}$ with codeword multiplicity $g^{(\mu)}$ and $\mathcal{V}_{\mathsf{Batch}}$ checks the padding using the procedure in Remark 4.13 (w.r.t. security parameter λ).
   (b) If $\mathcal{V}_{\mathsf{amplify}}$ rejects, then $\mathcal{V}_{\mathsf{Batch}}$ immediately rejects and halts. Otherwise, the protocol outputs $(w^{(i)})'$ and $\{(x_j^{(i+1)})'\}_{j \in S^{(i)}}$. Denote by $w^{(i+1)} = (w^{(i)}, (w^{(i)})')$ and $x_j^{(i+1)} = (x_j^{(i)}, (x_j^{(i)})')$, for every $j \in S^{(i)}$.
   (c) $\mathcal{V}_{\mathsf{Batch}}$ selects at random a subset $S^{(i+1)} \subseteq S^{(i)}$, of size $|S^{(i+1)}| = \frac{2|S^{(i)}|}{d} \cdot \lambda = \frac{|S^{(i)}|}{k^{1/\mu}}$, and sends $S^{(i+1)}$ to $\mathcal{P}_{\mathsf{Batch}}$.[a]
3. $\mathcal{P}_{\mathsf{Batch}}$ and $\mathcal{V}_{\mathsf{Batch}}$ run in parallel $|S^{(\mu)}|$ copies of the protocol $(\mathcal{P}_{\mathcal{L}^{(\mu)}}, \mathcal{V}_{\mathcal{L}^{(\mu)}})$, where the input for protocol $j \in S^{(\mu)}$ is $(w^{(\mu)}, x_j^{(\mu)})$. (Note that by the loop's invariant, $\mathcal{V}_{\mathsf{Batch}}$ has oracle access to $\mathsf{LDE}(x_j^{(\mu)})$, for every $j \in S^{(\mu)}$.
   The verifier $\mathcal{V}_{\mathsf{Batch}}$ accepts if $\mathcal{V}_{\mathcal{L}^{(\mu)}}$ accepts in all of these protocols and otherwise it rejects.

[a] To see that the loop's invariant is maintained and in particular that $\mathcal{V}_{\mathsf{Batch}}$ has oracle access to $\mathsf{LDE}(x_j^{(i+1)})$, for every $j \in S^{(i+1)}$, observe that each $x_j^{(i+1)}$ is composed of x_j concatenated with additional strings that the verifier has explicitly generated in previous iterations. Thus, each query to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_j^{(i+1)})$ can be simulated by a single oracle query to $\mathsf{LDE}(x_j)$ which in turn can be simulated by an oracle query to the implicit input $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x_1, \ldots, x_k)$ (see Proposition 3.8).

Figure 3: Batch Verification Protocol

Prescribed Completeness for $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$. Let $x_1, \ldots, x_k \in \{0,1\}^n$ be implicit inputs and let w be an explicit input.

Consider first the case that $(w, x_j) \in \mathcal{L}$ for every $j \in [k]$ (recall that in this case the verifier needs to accept). Consider the $x^{(i)}$'s and $w^{(i)}$'s generated in the interaction of $\mathcal{V}_{\mathsf{Batch}}$ with the *prescribed prover* $\mathcal{P}_{\mathsf{Batch}}$. By induction on i and using the prescribed completeness of $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$ for every $i \in \{0, \ldots, \mu)$ and $j \in S^{(i)}$ it holds that $(w^{(i)}, x_j^{(i)}) \in \mathcal{L}^{(i)}$. In particular $(w^{(\mu)}, x_j^{(\mu)}) \in \mathcal{L}^{(\mu)}$ for every $j \in S^{(\mu)}$. Now, by the prescribed completeness of $(\mathcal{P}_{\mathcal{L}^{(\mu)}}, \mathcal{V}_{\mathcal{L}^{(\mu)}})$ (which can by shown by induction using Lemma 6.3) in Step 3 the verifier $\mathcal{V}_{\mathsf{Batch}}$ accepts.

On the other hand, if there exists $j \in [k]$ such that $(w, x_j) \notin \mathcal{L}$ then, by Lemma 6.3 the verifier $\mathcal{V}_{\mathsf{Batch}}$ rejects already in the first iteration (i.e., when $i = 0$).

The fact that $\mathcal{V}_{\mathsf{Batch}}$ only makes input-oblivious queries to its transcript follows directly from the fact that $\mathcal{V}_{\mathsf{amplify}}$ makes input-oblivious queries and $\mathcal{V}_{\mathcal{L}^{(\mu)}}$ makes input-oblivious queries (see Lemma 6.3) as well as the fact the randomly chosen sets $S^{(i)}$ are part of $\mathcal{V}_{\mathsf{Batch}}$'s random string.

Unambiguity for $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$. Let $x_1, \ldots, x_k \in \{0,1\}^n$ and $w \in \{0,1\}^{n_{\mathsf{explicit}}}$, let $\widetilde{\mathcal{P}}_{\mathsf{Batch}}$ be a $(g^{(\mu)}, \mathbb{H}, \mathbb{F})$-encoded cheating prover. There are two cases to consider. Either the prover first deviates in one of the μ iterations of the deviation amplification protocol (i.e., Step 2 in the protocol) or its first deviation is in Step 3.

We first observe that in the second case (i.e., the first deviation is in Step 3) by the fact that $(\mathcal{P}_{\mathcal{L}^{(\mu)}}, \mathcal{V}_{\mathcal{L}^{(\mu)}})$ is an unambiguous PCIP (see Lemma 6.10) with unambiguity error $\varepsilon^{(\mu)} = \varepsilon + \mu \cdot 2^{-\lambda}$, with all but $\varepsilon + \mu \cdot 2^{-\lambda}$ probability, the verifier $\mathcal{V}_{\mathsf{Batch}}$ rejects. Hence we can focus on the first case.

Fix $i^* \in \{0, \ldots, \mu - 1\}$ and random coins for the for the verifier in the first $i^* - 1$ iterations of Step 2 such that $\widetilde{\mathcal{P}}_{\mathsf{Batch}}$ first deviates from the protocol in the $i^{*\text{th}}$ iteration.

Claim 6.10.1. *For every $i \geq i^*$, with probability $1 - i \cdot (\varepsilon + 2^{-\lambda+1})$, at the end of iteration i either the verifier $\mathcal{V}_{\mathsf{Batch}}$ has already rejected or it produces a set $S^{(i+1)}$ and strings $w^{(i+1)}$ and $(x_j^{(i+1)})_{j \in S^{(i+1)}}$ such that there exists $j \in S^{(i+1)}$ such that $(w^{(i+1}, x_j^{(i+1)}) \notin \mathcal{L}^{(i+1)}$.*

Proof. The proof is by induction over the value of i. For the base case (i.e., $i = i^*$), since $\tilde{P}_{\mathsf{Batch}}$ deviates in the $(i^*)^{\text{th}}$ iteration, by the unambiguity guarantee of the deviation amplification protocol, with probability $1 - \varepsilon - 2^{-\lambda}$ over the coin tosses of $\mathcal{V}_{\mathsf{amplify}}$ within the $i^{*\text{th}}$ iteration, either $\mathcal{V}_{\mathsf{amplify}}$ rejects (in which case also $\mathcal{V}_{\mathsf{Batch}}$ rejects) or there exists a set $J \subset S^{(i^*)}$ of size $|J| = d/2$ such that $\forall j \in J$, $\left(w^{(i^*+1)}, x_j^{(i^*+1)}\right) \notin \mathcal{L}^{(i^*+1)}$.

Hence, in Step 2c, the verifier $\mathcal{V}_{\mathsf{Batch}}$ selects a subset $S^{(i+1)} \subseteq S^{(i)}$ such that $S^{(i^*+1)} \cap J \neq \emptyset$, with probability

$$1 - \left(1 - \frac{|J|}{|S^{(i^*)}|}\right)^{|S^{(i^*+1)}|} = 1 - \left(1 - \frac{d}{2|S^{(i^*)}|}\right)^{\frac{2|S^{(i^*)}|}{d} \cdot \lambda} \geq 1 - 2^{-\lambda},$$

and the base case follows.

For $i > i^*$, by the induction hypothesis, with probability $1 - (i-1) \cdot (\varepsilon + 2^{(-\lambda+1)})$ over the coins of the verifier in the first $i - 1$ iterations, either $\mathcal{V}_{\mathsf{Batch}}$ rejects by the end of round $i - 1$ or there exists $j \in S^{(i)}$ such that $(w^{(i}, x_j^{(i)}) \notin \mathcal{L}^{(i)}$. If in iteration i the cheating prover follows the prescribed strategy then, by the prescribed completeness of $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$ the verifier rejects. Otherwise (i.e., the cheating prover deviates in the i^{th} iteration), by the unambiguity guarantee of Lemma 6.3,

with probability $1-\varepsilon-2^{-\lambda}$ either $\mathcal{V}_{\mathsf{Batch}}$ rejects or there exists a set $J \subset S^{(i)}$ of size $|J| = d/2$ such that $\forall j \in J$, $\left(w^{(i+1)}, x_j^{(i+1)}\right) \notin \mathcal{L}^{(i+1)}$. Hence, similar to the analysis in the base case, with probability $1-2^{-\lambda}$ over the choice of $S^{(i+1)}$ there exists $j \in S^{(i+1)}$ such that $\left(w^{(i+1)}, x_j^{(i+1)}\right) \notin \mathcal{L}^{(i+1)}$. Thus, by a union bound, with probability $1-(i-1)\cdot(\varepsilon+2^{-\lambda+1})-\varepsilon-2\cdot 2^{-\lambda} = 1-i\cdot(\varepsilon+2^{-\lambda+1})$ the verifier either rejects or there exists $j \in S^{(i+1)}$ such that $(w^{(i+1}, x_j^{(i+1)}) \notin \mathcal{L}^{(i+1)}$. □

Thus, with probability $1-\mu\cdot(\varepsilon+2^{-\lambda+1})$ either $\mathcal{V}_{\mathsf{Batch}}$ rejects in the first step (i.e., in one of the μ iterations) or it produces a set $S^{(\mu)}$ and strings $w^{(\mu)}$ and $(x_j^{(\mu)})_{j\in S^{(\mu)}}$ such that there exists some $j^* \in S^{(\mu)}$ such that $(w^{(\mu}, x_{j^*}^{(\mu)}) \notin \mathcal{L}^{(\mu)}$. Assume that the latter holds.

In Step 3 we run the protocol $(\mathcal{P}_{\mathcal{L}^{(\mu)}}, \mathcal{V}_{\mathcal{L}^{(\mu)}})$ on $(w^{(\mu)}, x_{j^*}^{(\mu)})$. If the prover $\tilde{P}_{\mathsf{Batch}}$ does not deviate within that protocol then, by its prescribed completeness, the verifier rejects with probability 1. If the prover does deviate then, by the protocol's unambiguity, with probability $1-\varepsilon^{(\mu)} = 1-\varepsilon-\mu\cdot 2^{-\lambda}$ the verifier rejects. By a union bound, the verifier rejects with probability at least $1-(\mu+1)\cdot\varepsilon-3\mu\cdot 2^{-\lambda} \le 1-(\mu+1)\cdot(\varepsilon+3\cdot 2^{-\lambda})$.

Complexity Measures for $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$. Each of the μ executions of the deviation amplification protocol $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$ within $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$ are protocols with parameters that are (upper bounded) as follows:

- Query complexity:

$$\begin{aligned} q_{\mathsf{final}} &\le q^{(\mu)}\cdot k^{(\mu)}\cdot \mathsf{poly}(\lambda,|\mathbb{H}|) + (k^{(\mu)})^2\cdot\mathsf{poly}(\ell^{(\mu)}, g^{(\mu)}, \lambda, |\mathbb{H}|) \\ &\le k^2\cdot q\cdot \mathsf{poly}(g)\cdot\big(\mathsf{poly}(\ell,\lambda,|\mathbb{H}|)\big)^{\mu}. \end{aligned}$$

- Rounds: $\ell_{\mathsf{final}} = \ell + 1$.
- Verifier Message Length: $b_{\mathsf{final}} \le \max\big(b, k\cdot\mathsf{poly}(|\mathbb{H}|, g, \ell^{\mu}, \lambda)\big)$.
- Prover Message Length: $a_{\mathsf{final}} \le d\cdot\log(k)\cdot\ell^{\mu}\cdot a$.
- Verifier Time:

$$\begin{aligned} \mathcal{V}\mathsf{time}_{\mathsf{final}} &\le k^{(\mu)}\cdot\mathcal{V}\mathsf{time}^{(\mu)} + q^{(\mu)}\cdot k^{(\mu)}\cdot\mathsf{poly}(\lambda,|\mathbb{H}|) + (k^{(\mu)})^2\cdot\mathsf{poly}(\ell^{(\mu)}, g^{(\mu)}, \lambda, |\mathbb{H}|) \\ &\le \mathcal{V}\mathsf{time}\cdot q\cdot k^2\cdot\big(\mathsf{poly}(\ell,\lambda,|\mathbb{H}|)\big)^{\mu}\cdot\mathsf{poly}(g). \end{aligned}$$

- Prover Time:

$$\begin{aligned} \mathcal{P}\mathsf{time}_{\mathsf{final}} &= 2k^{(\mu)}\cdot\mathcal{P}\mathsf{time}^{(\mu)} + k^{(\mu)}\cdot\mathcal{V}\mathsf{time}^{(\mu)} + q^{(\mu)}\cdot k^{(\mu)}\cdot\mathsf{poly}(\lambda,|\mathbb{H}|) + (k^{(\mu)})^2\cdot\mathsf{poly}(\ell^{(\mu)}, g^{(\mu)}, \lambda, |\mathbb{H}|) \\ &\le 2k\cdot\ell^{\mu}\cdot\mathcal{P}\mathsf{time} + \mathcal{V}\mathsf{time}\cdot q\cdot k^2\cdot\mathsf{poly}(g)\big(\mathsf{poly}(\ell,\lambda,|\mathbb{H}|)\big)^{\mu+1}. \end{aligned}$$

- Codeword Multiplicity: $g_{\mathsf{final}} = d\cdot\log(k)\cdot\ell^{\mu}\cdot g$.

The entire protocol $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$ consists of μ executions of $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$ with parameters upper bounded as above, followed by $|S^{(\mu)}| = O(1)$ parallel executions of $(\mathcal{P}_{\mathcal{L}^{(\mu)}}, \mathcal{V}_{\mathcal{L}^{(\mu)}})$. In between executions of $(\mathcal{P}_{\mathsf{amplify}}, \mathcal{V}_{\mathsf{amplify}})$ the verifier $\mathcal{V}_{\mathsf{Batch}}$ also specifies the sub-sampled sets to $\mathcal{P}_{\mathsf{Batch}}$ (each of size $|S^{(i)}| \leq k$) and checks the padding (this requires $\ell_{\mathsf{final}} \cdot g_{\mathsf{final}} \cdot \mathsf{poly}(|\mathbb{H}|)$ queries per iteration).

Thus, $(\mathcal{P}_{\mathsf{Batch}}, \mathcal{V}_{\mathsf{Batch}})$ is an ε_{A}-unambiguous $(q_{\mathsf{A}}, \ell_{\mathsf{A}}, b_{\mathsf{A}}, a_{\mathsf{A}}, \mathcal{P}\mathsf{time}_{\mathsf{A}}, \mathcal{V}\mathsf{time}_{\mathsf{A}})$-PCIP for $\mathcal{L}^{\otimes k}$ w.r.t. $(g_{\mathsf{A}}, \mathbb{H}, \mathbb{F})$-encoded provers, where:

- $\varepsilon_{\mathsf{A}} = (\mu+1) \cdot (\varepsilon + 3 \cdot 2^{-\lambda})$ (this was shown in the analysis of the unambiguity above).
- $q_{\mathsf{A}} \leq \mu \cdot \big(q_{\mathsf{final}} + \ell_{\mathsf{final}} \cdot g_{\mathsf{final}} \cdot \mathsf{poly}(|\mathbb{H}|, \lambda)\big) + O(q^{(\mu)}) = k^2 \cdot q \cdot \mathsf{poly}(g) \cdot \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{\mu+1}$.
- $\ell_{\mathsf{A}} = \mu \cdot (\ell_{\mathsf{final}} + 1) + \ell^{(\mu)} = \ell \cdot (\mu+1) + 2\mu$.
- $b_{\mathsf{A}} = \max(b_{\mathsf{final}}, b^{(\mu)}, k) = \max\big(b, k \cdot \mathsf{poly}(|\mathbb{H}|, \ell^{\mu}, g, \lambda)\big)$.
- $a_{\mathsf{A}} = \max(a_{\mathsf{final}}, a^{(\mu)}) = d \cdot \log(k) \cdot \ell^{\mu} \cdot a = a \cdot 2k^{1/\mu} \cdot \log(k) \cdot \lambda \cdot \ell^{\mu}$.
- $\mathcal{P}\mathsf{time}_{\mathsf{A}} = \mu \cdot \mathcal{P}\mathsf{time}_{\mathsf{final}} + O(\mathcal{P}\mathsf{time}^{(\mu)}) \leq 2k \cdot \mu \cdot \ell^{\mu} \cdot \mathcal{P}\mathsf{time} + \mathcal{V}\mathsf{time} \cdot q \cdot k^2 \cdot \mathsf{poly}(g)\big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{\mu+1}$.
- $\mathcal{V}\mathsf{time}_{\mathsf{A}} = \mu \cdot \big(\mathcal{V}\mathsf{time}_{\mathsf{final}} + \ell_{\mathsf{final}} \cdot g_{\mathsf{final}} \cdot \mathsf{poly}(|\mathbb{H}|, \lambda)\big) + O(\mathcal{V}\mathsf{time}_{\mu}) = \mathcal{V}\mathsf{time} \cdot q \cdot k^2 \cdot \mathsf{poly}(g) \cdot \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{\mu+1}$.
- $g_{\mathsf{A}} = \max(g_{\mathsf{final}}, g^{(\mu)}) = d \cdot \log(k) \cdot \ell^{\mu} g = g \cdot 2k^{1/\mu} \cdot \log(k) \cdot \lambda \cdot \ell^{\mu}$.

(where throughout we used the fact that $\mu \leq \log(a) \leq |\mathbb{H}|$).

This completes the proof of Lemma 6.1 (modulo the proofs of Lemma 6.5 and Proposition 6.6 which appear next).

6.4 Proofs of Lemma 6.5 and Proposition 6.6

6.4.1 Proof of Lemma 6.5

Given as explicit input $w \in \{0,1\}^{n_{\mathsf{explicit}}}$, $\beta^{(1)}, \ldots, \beta^{(\ell)} \in \mathbb{F}^b$ and implicit input $x \in \{0,1\}^n$ and $\big(\tilde{\alpha}^{(1)}, \ldots, \tilde{\alpha}^{(\ell)}\big) \in (\mathbb{F}^a)^{\ell}$, the transcript tester $\mathcal{T}$ works as follows:

1. First $\mathcal{T}$ runs an individual degree test (see Lemma 3.5) on every one of the $g \cdot \ell$ alleged codewords in the transcript, w.r.t. individual degree $|\mathbb{H}| - 1$, proximity parameter $\frac{\delta}{g \cdot \ell}$ and soundness parameter λ. If any of these tests fail then $\mathcal{T}$ rejects.
2. $\mathcal{T}$ runs the verifier $\mathcal{V}_{\mathcal{L}}$ with explicit input $\big(w, \big(\beta^{(1)}, \ldots, \beta^{(\ell)}\big)\big)$ while forwarding input queries that $\mathcal{V}_{\mathcal{L}}$ makes x and emulating each transcript query $\xi \in [\ell \cdot a]$ that $\mathcal{V}_{\mathcal{L}}$ makes by running the self-correction procedure on the implicit input $\big(\tilde{\alpha}^{(1)}, \ldots, \tilde{\alpha}^{(\ell)}\big)$ at the point ξ, w.r.t. total degree $|\mathbb{H}| \cdot m$ and soundness parameter $(\lambda + \log(\ell \cdot a))$. The same random bits are used for all of the self-correction procedures.

Prescribed Completeness. Let $\tilde{\alpha}^{(1)},\ldots,\tilde{\alpha}^{(\ell)} \in \mathbb{F}^a$ such that each $\tilde{\alpha}^{(i)}$ is composed of g $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords. Since each one of the alleged $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords is indeed such a codeword, all the low degree tests pass and all the self-correction procedures simply return the relevant value in $\tilde{\alpha}^{(i)}$. Hence, by construction, $\mathcal{T}$ outputs $\mathcal{V}_{\mathcal{L}}\left(\left(w,\left(\beta^{(1)},\ldots,\beta^{(\ell)}\right)\right),\left(x,\left(\tilde{\alpha}^{(1)},\ldots,\tilde{\alpha}^{(\ell)}\right)\right)\right)$.

Since $\mathcal{V}_{\mathcal{L}}$ makes *input-oblivious* queries, for every two inputs $x_1,x_2 \in \{0,1\}^n$, every $w \in \{0,1\}^{n_{\mathsf{explicit}}}$, every $\beta^{(1)},\ldots,\beta^{(\ell)} \in \mathbb{F}^b$ and every random string, the queries that $\mathcal{V}_{\mathcal{L}}$ generates w.r.t. x_1 and x_2 are the same, when interacting with the *prescribed* prover.

The queries that the transcript tester are just low degree tests (which are totally independent of the input and the prover) and the local correction procedure at the foregoing points specified by $\mathcal{V}_{\mathcal{L}}$. Thus, the queries that $\mathcal{T}$ makes are input-oblivious.

Soundness. Let $x \in \{0,1\}^n$, $w \in \{0,1\}^{n_{\mathsf{explicit}}}$ $\beta^{(1)},\ldots,\beta^{(\ell)} \in \mathbb{F}^b$ and $\left(\tilde{\alpha}^{(1)},\ldots,\tilde{\alpha}^{(\ell)}\right) \in (\mathbb{F}^a)^\ell$ that are δ-far from the set

$$\left\{\left(\bar{\alpha}^{(1)},\ldots,\bar{\alpha}^{(\ell)}\right) \in ((\mathsf{LDE}_{\mathbb{F},\mathbb{H},m})^g)^\ell \;:\; \mathcal{V}_{\mathcal{L}}\left(\left(w,\left(\beta^{(1)},\ldots,\beta^{(\ell)}\right)\right),\left(\hat{x},\left(\bar{\alpha}^{(1)},\ldots,\bar{\alpha}^{(\ell)}\right)\right)\right)=1\right\}$$

Suppose first that one of the $\tilde{\alpha}^{(i)}$'s contains a string that is not $\frac{\delta}{\ell\cdot g}$ close to some individual degree $|\mathbb{H}|-1$ polynomial. Then, by Lemma 3.5, the low degree test for that string rejects with probability $1-2^{\lambda}$, in which case $\mathcal{T}$ rejects and we are done.

Hence, we assume that $\left(\tilde{\alpha}^{(1)},\ldots,\tilde{\alpha}^{(\ell)}\right)$ is δ-close to some $\left(\bar{\alpha}^{(1)},\ldots,\bar{\alpha}^{(\ell)}\right)$ such that each $\bar{\alpha}^{(i)}$ is composed of g $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}$ codewords. By our assumption on $\left(\tilde{\alpha}^{(1)},\ldots,\tilde{\alpha}^{(\ell)}\right)$, it must be the case that

$$\mathcal{V}_{\mathcal{L}}\left(\left(w,\left(\beta^{(1)},\ldots,\beta^{(\ell)}\right)\right),\left(\hat{x}_j,\left(\bar{\alpha}^{(1)},\ldots,\bar{\alpha}^{(\ell)}\right)\right)\right)=0. \tag{7}$$

When $\mathcal{T}$ emulates $\mathcal{V}_{\mathcal{L}}$, by the guarantee of the self-correction procedure (see Lemma 3.3), with probability $1-2^{\lambda+\log(\ell\cdot a)}$, each individual query that $\mathcal{V}_{\mathcal{L}}$ makes is answered according to $\left(\bar{\alpha}^{(1)},\ldots,\bar{\alpha}^{(\ell)}\right)$. Taking a union bound over all the queries, and noting that there are at most $\ell\cdot a$ queries to the transcipt, we have that with probability $1-2^{-\lambda}$ the tester $\mathcal{T}$ emulates $\mathcal{V}_{\mathcal{L}}$ while consistently answering according to $\left(\bar{\alpha}^{(1)},\ldots,\bar{\alpha}^{(\ell)}\right)$ and so, by Eq. (7), $\mathcal{T}$ rejects in Step 2.

Complexity Measures. The tester runs $\ell\cdot g$ individual degree tests, each w.r.t. individual degree $|\mathbb{H}|-1$, proximity parameter $\frac{\delta}{g\cdot\ell}$ and soundness parameter λ. By Lemma 3.5 this can be done in time $\ell\cdot g\cdot O(|\mathbb{H}|\cdot m\cdot g\cdot\ell\cdot\lambda\cdot 1/\delta)=\frac{1}{\delta}\cdot\mathsf{poly}(\ell,g,|\mathbb{H}|,\lambda)$ with similar query and randomness complexity.

As for the self-correction procedure, it is run on q points, each w.r.t. total degree $|\mathbb{H}|\cdot m$ and soundness parameter $\lambda+\log(\ell\cdot a)$. This takes time $q\cdot O(|\mathbb{H}|\cdot m\cdot(\lambda+\log(\ell\cdot a)))=q\cdot\mathsf{poly}(|\mathbb{H}|,\lambda)$ and a similar number of queries. Since we reuse the random bits for self-correction, the randomness complexity is $\mathsf{poly}(|\mathbb{H}|,\lambda)$.

Lastly, running $\mathcal{V}_{\mathcal{L}}$ (which is deterministic given $\beta^{(1)},\ldots,\beta^{(\ell)}$) takes time $\mathcal{V}\mathsf{time}$ and requires an additional q queries to the input x.

Overall we obtain running time $\mathcal{V}\mathsf{time}+q\cdot\mathsf{poly}(|\mathbb{H}|,\lambda)+\frac{1}{\delta}\cdot\mathsf{poly}(\ell,g,|\mathbb{H}|,\lambda)$, query complexity $q\cdot\mathsf{poly}(|\mathbb{H}|,\lambda)+\frac{1}{\delta}\cdot\mathsf{poly}(\ell,g,|\mathbb{H}|,\lambda)$ and randomness complexity $\frac{1}{\delta}\cdot\mathsf{poly}(\ell,g,|\mathbb{H}|,\lambda)$.

This completes the proof of Lemma 6.5.

6.4.2 Proof of Proposition 6.6

Recall that our goal is to construct an $\mathbb{F}$-linear systematic error-correcting code $C : \mathbb{F}^k \to \mathbb{F}^{k+d\log(k)}$ with absolute distance $\Omega(d)$. While the Reed Solomon code is such a code, it requires that the field size $\mathbb{F}$ be quite large (i.e., $|\mathbb{F}| > k$) which we cannot afford. Instead we turn to a slightly more elaborate construction, in which we basically employ the Reed Solomon code over an *extension field* of $\mathbb{F}$.

We turn to the proof of Proposition 6.6. Let $\mathbb{F}' = \mathbb{F}^{\log(k)}$, where we view $\mathbb{F}'$ as an extension field of $\mathbb{F}$. Let $C_{\mathbb{F}'} : (\mathbb{F}')^{k/\log(k)} \to (\mathbb{F}')^{k/\log(k)+d}$ be the Reed-Solomon code over $\mathbb{F}'$. Note that $C_{\mathbb{F}'}$ is a systematic $\mathbb{F}'$-linear code with distance $d+1$ and can be computed in time $k \cdot \mathsf{poly}(\log(|\mathbb{F}'|), \log(k)) = k \cdot \mathsf{poly}(\log(|\mathbb{F}|, \log(k)))$ (e.g., using FFT).

For $x \in \mathbb{F}^k$, we denote by $\bar{x} \in \mathbb{F}'^{k/\log(k)}$ the vector that is obtained by viewing x as an element of $(\mathbb{F}')^{k/\log(k)}$ in the natural way. Consider the function $C_{\mathbb{F}} : \mathbb{F}^k \to \mathbb{F}^{k+d\log(k)}$ defined as $C_{\mathbb{F}}(x) \stackrel{\text{def}}{=} \mathcal{C}_{\mathbb{F}'}(\bar{x})$. The code $C_{\mathbb{F}}$ is systematic, has distance $d+1$ and can be computed in time $k \cdot \mathsf{poly}(\log(|\mathbb{F}|), \log(k))$. It remains to be shown that it is $\mathbb{F}$-linear.

Using the linearity of $C_{\mathbb{F}'}$, for every $x, y \in \mathbb{F}^k$ it holds that

$$C_{\mathbb{F}}(x+y) = C_{\mathbb{F}'}\big(\overline{x+y}\big) = C_{\mathbb{F}'}(\bar{x}+\bar{y}) = C_{\mathbb{F}'}(\bar{x}) + C_{\mathbb{F}'}(\bar{y}) = C_{\mathbb{F}}(x) + C_{\mathbb{F}}(y).$$

To show that multiplication by a scalar preserves linearity, we will use the following claim:

Claim 6.10.2. *For every $\lambda \in \mathbb{F}$, there exists $\bar{\lambda} \in \mathbb{F}'$ such that for every $x \in \mathbb{F}^k$ it holds that $\lambda \cdot x = \bar{\lambda} \cdot \bar{x}$.*

Proof. Fix an irreducible polynomial Q over $\mathbb{F}$. We can view elements of $\mathbb{F}' = \mathbb{F}^{\log(k)}$ as degree $\log(k)-1$ polynomials over $\mathbb{F}$ where addition and multiplication of field elements corresponds to addition and multiplication modulo Q. We denote by $\bar{\lambda} \in \mathbb{F}'$ the constant (i.e., degree 0) polynomial which always outputs λ. Indeed, for every $x \in \mathbb{F}^k$ it holds that $\lambda \cdot x = \bar{\lambda} \cdot \bar{x}$. □

For every $\lambda \in \mathbb{F}$ and $x \in \mathbb{F}^k$ it holds that

$$C_{\mathbb{F}}(\lambda \cdot x) = C_{\mathbb{F}'}(\bar{\lambda} \cdot \bar{x}) = \bar{\lambda} \cdot C_{\mathbb{F}'}(\bar{x}) = \lambda \cdot C_{\mathbb{F}}(x)$$

where $\bar{\lambda}$ is the value defined in Claim 6.10.2.

This completes the proof of Proposition 6.6.

7 The Query-Reduction Transformation for PCIPs

In this section, we show a query-reduction transformation for unambiguous PCIPs w.r.t. encoded provers. This transformation takes a PCIP $(\mathcal{P}, \mathcal{V})$ and produces a new PCIP $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$, whose query complexity and verifier runtime are significantly smaller. The main cost is an increase in the communication, which grows (additively) by roughly $\mathsf{poly}(\mathcal{V}\mathsf{time})$. We mention that we use this transformation in two places: to reduce the amount of queries and verifier running time that the batch verification lemma (Lemma 8.1) incurs and to transform general unambiguous IPs to unambiguous PCIPs (see Proposition 4.15). We proceed with an overview of the transformation, which is inspired by the classical PCP of Babai *et al.* [BFLS91] (which has poly-logarithmically many queries).

The main idea is to "delegate" $\mathcal{V}$'s verification (queries and computation) to the untrusted prover $\mathcal{P}_{\mathsf{Qreduce}}$, and then verifying that this computation was performed correctly. Towards this, we consider a language $\mathcal{L}_{\mathcal{V}}$ over tuples (β, Q, ϕ), where β are $\mathcal{V}$'s random coins in the protocol $(\mathcal{P}, \mathcal{V})$, Q is a set of q queries that $\mathcal{V}$ could make (into the transcript and the input), and $\phi : Q \to \mathbb{F}$ are the values that $\mathcal{V}$ could read in the coordinates specified in Q. An input (β, Q, ϕ) is in $\mathcal{L}_{\mathcal{V}}$ if and only if the set Q is indeed the set of queries that $\mathcal{V}$ would make on randomness β, and if $\mathcal{V}$ would accept given the values in ϕ (and the randomness β). The main fact that we use is that $\mathcal{L}_{\mathcal{V}}$ is computable by a Turing Machine running in time $\mathcal{V}\mathsf{time}$ (namely by running the query and computation steps of the verifier $\mathcal{V}$).

The PCIP $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ proceeds as follows:

1. $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ run the original PCIP $(\mathcal{P}, \mathcal{V})$ to generate a communication transcript, but they stop short of making $\mathcal{V}$'s queries (to the transcript and the input) or running $\mathcal{V}$'s verification.

   In this execution of $(\mathcal{P}, \mathcal{V})$, let β be $\mathcal{V}$'s random coins, let Q be the set of queries (to the input and the transcript) that $\mathcal{V}$ makes with randomness β, and let $\phi : Q \to \mathbb{F}$ be the set of transcript and input values in the coordinates specified by Q.

2. The prescribed prover $\mathcal{P}_{\mathsf{Qreduce}}$ sends to $\mathcal{V}_{\mathsf{Qreduce}}$ a low-degree extension $\hat{v}$ of $v = (\beta, Q, \phi)$.

3. $\mathcal{V}_{\mathsf{Qreduce}}$ receives an LDE $\tilde{v}$. Let v' be the message encoded in $\tilde{v}$. $\mathcal{P}_{\mathsf{Qreduce}}$ and $\mathcal{V}_{\mathsf{Qreduce}}$ run a PCIP for verifying that $v' \in \mathcal{L}_{\mathcal{V}}$.

   For this, we use an unambiguous PCIP $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ w.r.t. encoded provers for any language that is computable by a Turing Machine in time T. This is a high-communication protocol, with communication complexity $\mathsf{poly}(T)$. Its main advantage is that it has a constant number of rounds ρ, and the query complexity and verification time are only roughly $T^{O(1/\rho)}$, i.e. much smaller than T.

   The PCIP $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ is described in Lemma 7.3 of Section 7.2. It uses algebraic techniques from the PCP and interactive-proof literature, most notably the interactive sumcheck protocol [LFKN92] (in particular, we need a constant-round version of this protocol). We show that the sumcheck protocol has unambiguous soundness.

4. $\mathcal{V}_{\mathsf{Qreduce}}$ verifies that $\tilde{v}$ sent by $\mathcal{P}_{\mathsf{Qreduce}}$ is consistent with the actual transcript, the randomness and the input in the execution of $(\mathcal{P}, \mathcal{V})$. This is done using another ρ-round interactive sumcheck protocol, where the verifier's query complexity and computation are roughly $(q \cdot T)^{O(1/\rho)}$.

5. $\mathcal{V}_{\mathsf{Qreduce}}$ accepts if and only if the above two tests are successful.

The remainder of this section is organized as follows. We describe the sumcheck protocol and prove that it is unambiguous in Section 7.1. In Section 7.2 we present the PCIP $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ described above. Finally, we construct the transformed PCIP $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ and prove Lemma 8.2 in Section 7.3.

7.1 The Sumcheck Protocol

We use (a slight variant of) the sumcheck protocol of [LFKN92] for verifying the sum of a low-degree polynomial P over all inputs in $\mathbb{H}^m$. The main new fact that we prove about this canonical protocol is that it has *unambiguous* soundness. We provide a generalized presentation that allows for a trade-off between the number of rounds and the total amount of communication.[20]

Following [BFLS91], we view the verifier in the sumcheck protocol as not having any access to the polynomial P. Rather, at the end of the protocol, the verifier outputs a single point $\mathbf{r} \in \mathbb{F}^m$, and a corresponding value $\nu \in \mathbb{F}$. Loosely speaking, the case that the polynomial at the point $\mathbf{r}$ equals to (resp., differs from) ν corresponds to accepting (resp., rejecting).

Lemma 7.1 (The Sumcheck Protocol (cf.,[LFKN92]))**.** *Fix a constructible ensemble of fields $\mathbb{H} = (\mathbb{H}_n)_{n\in\mathbb{N}}$, a constructible ensemble of extension fields $\mathbb{F} = (\mathbb{F}_n)_{n\in\mathbb{N}}$ (i.e., $\mathbb{F}_n \supseteq \mathbb{H}_n$ is an extension field of $\mathbb{H}_n$) such that $|\mathbb{H}|_n \geq \log(|\mathbb{F}_n|)$.*

For every $m = m(n) \geq 1$ and round parameter $\rho = \rho(n) \in [m]$, where ρ divides m, there exists a $(\ell_{\mathsf{S}} = \rho, a_{\mathsf{S}}, b_{\mathsf{S}}, \mathcal{P}\mathsf{time}_{\mathsf{S}}, \mathcal{V}\mathsf{time}_{\mathsf{S}})$-interactive protocol $(\mathcal{P}_{\mathsf{sumchk}}, \mathcal{V}_{\mathsf{sumchk}})$ as follows.

The verifier $\mathcal{V}_{\mathsf{sumchk}}$ gets as input $n \in \mathbb{N}$. The prover $\mathcal{P}_{\mathsf{sumchk}}$ gets as input a polynomial $P\colon \mathbb{F}^m \to \mathbb{F}$ of individual degree $t = t(n) \in [|\mathbb{H}|, |\mathbb{F}| - 1]$. At the end of the interaction $\mathcal{V}_{\mathsf{sumchk}}$ either rejects or outputs a point $\mathbf{r} \in \mathbb{F}^m$, which depends only on $\mathcal{V}_{\mathsf{sumchk}}$'s coin tosses, and a value $v \in \mathbb{F}$ such that:

- **Prescribed Completeness:** *If $\sum_{z\in\mathbb{H}^m} P(z) = 0$, then $(\mathcal{P}_{\mathsf{sumchk}}(P, n), \mathcal{V}_{\mathsf{sumchk}}(n))$ outputs $(\mathbf{r}, v)$ such that $P(\mathbf{r}) = v$. Otherwise, $(\mathcal{P}_{\mathsf{sumchk}}(P), \mathcal{V}_{\mathsf{sumchk}}(n))$ rejects.*
- **Unambiguity:** *For every cheating prover $\widetilde{\mathcal{P}}_{\mathsf{sumchk}}$, for every round $i^* \in [\rho]$, and for every choice of $\mathcal{V}_{\mathsf{sumchk}}$'s coins before round i^*, if $\widetilde{P}_{\mathsf{sumchk}}$ first deviates from $(\mathcal{P}_{\mathsf{sumchk}}, \mathcal{V}_{\mathsf{sumchk}})$ in round i^*, then, with probability $(1 - \frac{t\cdot m}{|\mathbb{F}|})$, either $\mathcal{V}_{\mathsf{sumchk}}$ rejects or $P(\mathbf{r}) \neq v$. The probability is (only) over $\mathcal{V}_{\mathsf{sumchk}}$'s coin tosses in rounds $i^*, \ldots, \ell$.*

Finally, the protocol has:

- $a_{\mathsf{S}} = \big((t+1)^{m/\rho} \cdot \log|\mathbb{F}|\big)$
- $b_{\mathsf{S}} = ((m/\rho) \cdot \log|\mathbb{F}|)$
- $\mathcal{P}\mathsf{time}_{\mathsf{S}} = \big(|\mathbb{H}|^m \cdot t^{O(m/\rho)} \cdot \mathsf{poly}(m, \log|\mathbb{F}|)\big)$
- $\mathcal{V}\mathsf{time}_{\mathsf{S}} = \big(t^{O(m/\rho)} \cdot \mathsf{poly}(m, \log|\mathbb{F}|)\big)$

Remark 7.2 (Sumcheck as a PCIP)**.** *We use the Sumcheck protocol in the context of* PCIP*s $(\mathcal{P}, \mathcal{V})$, by having the* PCIP *prover and verifier run $(\mathcal{P}_{\mathsf{sumchk}}, \mathcal{V}_{\mathsf{sumchk}})$ as a sub-protocol. When this is the case, we want to bound the verifier $\mathcal{V}$'s query complexity in the sumcheck execution. We do so by simply having the* PCIP *verifier explicitly query every bit of the messages sent by $\mathcal{P}_{\mathsf{sumchk}}$. This gives query complexity:*

$$q_{\mathsf{S}} = (\rho \cdot (t+1)^{m/\rho} \cdot \log|\mathbb{F}|).$$

Since the verifier reads every bit of every message, we get that the protocol makes input-oblivious queries.

We note that the Sumcheck messages might be smaller than the messages sent in the PCIP *$(\mathcal{P}, \mathcal{V})$, which can be handled using Remark 4.13.*

[20] One way of viewing this tradeoff is as applying the sumcheck protocol for tensor codes of Meir [Mei13] to a different bases code (composed of lower dimensional tensors).

Proof of Lemma 7.1. Our protocol is a variant of the classical sumcheck protocol, in which the prover sends multivariate polynomials at each round (rather than univariate as in the classical case). We use this variant since it allows for flexibility in the number of rounds in the protocol (in contrast, in the classical protocol, the number of rounds is always equal to the dimension of the polynomial).

Throughout the proof we will often evaluate a polynomial $P : \mathbb{F}^m \to \mathbb{F}$ on points in $(\mathbb{F}^{m/\rho})^\rho$, by viewing such points as elements in $\mathbb{F}^m$ in the natural way. The (multi-variate) sumcheck protocol is described in Fig. 4.

The (multi-variate) Sumcheck Protocol:

Parameters: $\mathbb{F}$ (field), m (dimension), t (individual degree), $\mathbb{H} \subset \mathbb{F}$ and $\rho \in [m]$.

Prover's Input: an individual degree t polynomial $P : \mathbb{F}^m \to \mathbb{F}$, specified by its $(t+1)^m$ coefficients.

1. Let $\nu^{(0)} \stackrel{\text{def}}{=} 0$.
2. For $i \leftarrow 1, \ldots, \rho$:
   (at the beginning of round i both $\mathcal{V}_{\mathsf{sumchk}}$ and $\mathcal{P}_{\mathsf{sumchk}}$ know $r^{(1)}, \ldots, r^{(i-1)}$ and $\nu^{(0)}, \ldots, \nu^{(i-1)}$)
   (a) $\mathcal{P}_{\mathsf{sumchk}}$ sends to $\mathcal{V}$ the individual degree t polynomial $P^{(i)} : \mathbb{F}^{m/\rho} \to \mathbb{F}$ on m/ρ variables (by specifying its $(t+1)^{m/\rho}$ coefficients) defined as:
   $$P^{(i)}(z) \stackrel{\text{def}}{=} \sum_{x_{i+1},\ldots,x_\rho \in \mathbb{H}^{m/\rho}} P\left(r^{(1)}, \ldots, r^{(i-1)}, z, x_{i+1}, \ldots, x_\rho\right).$$
   (b) $\mathcal{V}_{\mathsf{sumchk}}$ receives an individual degree t polynomial $\widetilde{P}^{(i)} : \mathbb{F}^{m/\rho} \to \mathbb{F}$ (which is allegedly equal to $P^{(i)}$). $\mathcal{V}_{\mathsf{sumchk}}$ checks that $\sum_{z \in \mathbb{H}^{m/\rho}} \widetilde{P}_i(z) = \nu^{(i-1)}$ and rejects otherwise.
   (c) $\mathcal{V}$ selects uniformly at random $r^{(i)} \in_R \mathbb{F}^{m/\rho}$, sets $\nu^{(i)} \stackrel{\text{def}}{=} \widetilde{P}_i(r^{(i)})$, and sends $r^{(i)}$ to $\mathcal{P}_{\mathsf{sumchk}}$.
3. $\mathcal{V}$ outputs the point $\mathbf{r} = (r^{(1)}, \ldots, r^{(\rho)}) \in (\mathbb{F}^{m/\rho})^\rho$ (viewed as a point in $\mathbb{F}^m$) and the value $\nu = \nu^{(\rho)}$.

Figure 4: The Sumcheck Protocol

The running times and communication patterns of $\mathcal{P}_{\mathsf{sumchk}}$ and $\mathcal{V}_{\mathsf{sumchk}}$ can be readily verified. We proceed to show that completeness and unambiguity hold.

Prescribed Completeness. Let $P : \mathbb{F}^m \to \mathbb{F}$ be an individual degree t polynomial. Suppose first that $\sum_{x \in \mathbb{H}^m} P(x) \neq 0$. Then, in the first round $\mathcal{P}_{\mathsf{sumchk}}$ sends to $\mathcal{V}_{\mathsf{sumchk}}$ the polynomial $\widetilde{P}^{(1)} = P^{(1)}(z)$. Hence,

$$\sum_{z \in \mathbb{H}^{m/\rho}} \widetilde{P}^{(1)}(z) = \sum_{z \in \mathbb{H}^{m/\rho}} P^{(1)}(z) = \sum_{z \in \mathbb{H}^{m/\rho}} \sum_{x_2,\ldots,x_\rho \in \mathbb{H}^{m/\rho}} P(z, x_2, \ldots, x_\rho) = \sum_{x \in \mathbb{H}^m} P(x) \neq 0 = \nu^{(0)}$$

and so $\mathcal{V}_{\mathsf{sumchk}}$ rejects when checking that $\sum_{z \in \mathbb{H}^{m/\rho}} \widetilde{P}^{(1)}(z) = \nu^{(0)}$.

Now assume that $\sum_{x \in \mathbb{H}^m} P(x) = 0$. In this case, at every round $i \in [\rho]$, the prover $\mathcal{P}_{\mathsf{sumchk}}$

sends the polynomial $\widetilde{P}^{(i)} = P^{(i)}$. Hence, for every $i \in [\rho]$:

$$\begin{aligned}
\sum_{z \in \mathbb{H}^{m/\rho}} \widetilde{P}^{(i)}(z) &= \sum_{z \in \mathbb{H}^{m/\rho}} P^{(i)}(z) \\
&= \sum_{z \in \mathbb{H}^{m/\rho}} \sum_{x_{i+1},\ldots,x_\rho \in \mathbb{H}^{m/\rho}} P(r^{(1)}, \ldots, r^{(i-1)}, z, x_{i+1}, \ldots, x_m) \\
&= P^{(i-1)}(r^{(i-1)}) \\
&= \widetilde{P}^{(i-1)}(r^{(i-1)}) \\
&= \nu^{(i-1)}
\end{aligned}$$

and so all of $\mathcal{V}_{\mathsf{sumchk}}$'s checks pass. At the end of the protocol $\mathcal{V}_{\mathsf{sumchk}}$ outputs $\mathbf{r} = (r^{(1)}, \ldots, r^{(\rho)})$ and $\nu = \nu^{(\rho)} = P^{(\rho)}(r^{(\rho)}) = P\left(r^{(1)}, \ldots, r^{(\rho)}\right)$ as required.

Unambiguity. Let $P : \mathbb{F}^m \to \mathbb{F}$ be an individual degree t polynomial. Let $\widetilde{\mathcal{P}}_{\mathsf{sumchk}}$ be a cheating prover. Let $i^* \in [\rho]$ and $r^{(1)}, \ldots, r^{(i^*-1)} \in \mathbb{F}^{m/\rho}$ such that $\widetilde{\mathcal{P}}$ first deviates from $(\mathcal{P}_{\mathsf{sumchk}}, \mathcal{V}_{\mathsf{sumchk}})$ at round i^*. That is, $\widetilde{P}^{(i)} \equiv P^{(i)}$ for every $i < i^*$, but $\widetilde{P}^{(i^*)} \not\equiv P^{(i^*)}$, where $P^{(i)} = \mathcal{P}_{\mathsf{sumchk}}\left(P, i, \left(r^{(1)}, \ldots, r^{(i-1)}\right)\right)$ and $\widetilde{P}^{(i)} = \widetilde{\mathcal{P}}_{\mathsf{sumchk}}\left(P, i, \left(r^{(1)}, \ldots, r^{(i-1)}\right)\right)$.

Claim 7.2.1. *Let $i \in \{i^*, \ldots, \rho - 1\}$ and $r^{(i^*)}, \ldots, r^{(i-1)} \in \mathbb{F}$ and suppose that $\widetilde{P}^{(i)} \not\equiv P^{(i)}$. Then, with probability at least $1 - \frac{t \cdot (m/\rho)}{|\mathbb{F}|}$ over the choice of $r^{(i)}$, either $\mathcal{V}_{\mathsf{sumchk}}$ rejects or $\widetilde{P}^{(i+1)} \not\equiv P^{(i+1)}$.*

Proof. Since $\mathcal{P}^{(i)}$ and $\widetilde{P}^{(i)}$ have *total* degree at most $t \cdot (m/\rho)$, by the Schwartz-Zippel Lemma (Lemma 3.1), with probability $1 - \frac{t \cdot (m/\rho)}{|\mathbb{F}|}$ over the choice of $r^{(i)} \in \mathbb{F}^{m/\rho}$ it holds that $P^{(i)}\left(r^{(i)}\right) \neq \widetilde{P}^{(i)}\left(r^{(i)}\right)$. Suppose that this is the case and yet $\widetilde{P}^{(i+1)} \equiv P^{(i+1)}$. Then,

$$\sum_{z \in \mathbb{H}^{m/\rho}} \widetilde{P}^{(i+1)}(z) = \sum_{z \in \mathbb{H}^{m/\rho}} P^{(i+1)}(z) = P^{(i)}\left(r^{(i)}\right) \neq \widetilde{P}^{(i)}\left(r^{(i)}\right) = \nu^{(i)}$$

and so $\mathcal{V}_{\mathsf{sumchk}}$ rejects when checking that $\sum_{z \in \mathbb{H}^{m/\rho}} \widetilde{P}^{(i+1)}(z) = \nu^{(i)}$. □

By Claim 7.2.1 and taking a union bound over every $i \in \{i^*, \ldots, \rho - 1\}$, with probability at least $1 - (\rho - 1) \cdot \frac{t \cdot (m/\rho)}{|\mathbb{F}|}$, over the choice of $r_{i^*}, \ldots, r_{\rho-1} \in \mathbb{F}^{m/\rho}$, either $\mathcal{V}_{\mathsf{sumchk}}$ rejects or $\widetilde{P}^{(\rho)} \not\equiv P^{\rho}$. If $\widetilde{P}^{(\rho)} \not\equiv P^{\rho}$, then, by the Schwartz-Zippel Lemma, with probability at least $1 - \frac{t \cdot (m/\rho)}{|\mathbb{F}|}$ over the choice of $r^{(\rho)}$, it holds that

$$\widetilde{P}^{(\rho)}\left(r^{\rho}\right) \neq P^{\rho}\left(r^{(\rho)}\right) = P\left(r^{(1)}, \ldots, r^{(\rho)}\right).$$

Hence, by another application of the union bound, with probability at least $1 - \rho \cdot \frac{t \cdot (m/\rho)}{|\mathbb{F}|} = 1 - \frac{t \cdot m}{|\mathbb{F}|}$, either $\mathcal{V}_{\mathsf{sumchk}}$ rejects or $\nu = \nu^{(\rho)} = \widetilde{P}^{(\rho)}\left(r^{\rho}\right) \neq P\left(r^{(1)}, \ldots, r^{(\rho)}\right) = P(\mathbf{r})$. □

7.2 Unambiguous PCIP for T-time w.r.t. Encoded Provers

Lemma 7.3 (Unambiguous PCIP for Ttime w.r.t. Encoded Provers)**.** *Let $\mathcal{L}$ be a language computable by a $T = T(n)$-time Turing Machine. Let $\rho = \rho(n)$ be a round parameter. Take $\mathbb{H}$ and $\mathbb{F}$ to be constructible field ensembles where $|\mathbb{F}| \leq \mathsf{poly}(|\mathbb{H}|)$ and $\log(\max(T, n)) \leq |\mathbb{H}| \leq T^{1/\rho}$.*

There exists a protocol $(\mathcal{P}^{\mathcal{L}}_{\mathsf{Ttime}}, \mathcal{V}^{\mathcal{L}}_{\mathsf{Ttime}})$ that is an ϵ_{T}-unambiguous $(q_{\mathsf{T}}, \ell_{\mathsf{T}}, a_{\mathsf{T}}, b_{\mathsf{T}}, \mathcal{P}\mathsf{time}_{\mathsf{T}}, \mathcal{V}\mathsf{time}_{\mathsf{T}})$-PCIP for the language $\mathcal{L}$ w.r.t. $(g_{\mathsf{T}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries. Where:

- $q_{\mathsf{T}} = \mathsf{poly}(T)^{1/\rho}$
- $\ell_{\mathsf{T}} = (\rho + 1)$
- $a_{\mathsf{T}} = \mathsf{poly}(T)$
- $b_{\mathsf{T}} = O(|\mathbb{H}| \cdot \log |\mathbb{F}|)$
- $\mathcal{P}\mathsf{time}_{\mathsf{T}} = \mathsf{poly}(T)$
- $\mathcal{V}\mathsf{time}_{\mathsf{T}} = \mathsf{poly}(T)^{1/\rho}$
- $\epsilon_{\mathsf{T}} = \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}$
- $g_{\mathsf{T}} = 1$.

Proof. For ease of notation, denote $\mathcal{P}_{\mathsf{Ttime}} = \mathcal{P}^{\mathcal{L}}_{\mathsf{Ttime}}$ and $\mathcal{V}_{\mathsf{Ttime}} = \mathcal{V}^{\mathcal{L}}_{\mathsf{Ttime}}$. Let $\mathcal{M}$ be the T-time and S-space Turing Machine that decides $\mathcal{L}$. In the PCIP $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$, the prover first sends to the verifier the low-degree extension $\hat{y}$ of the tableau y of M's computation on input x. This tableau is *unique*, and its low-degree extension can be computed in time $\mathsf{poly}(T)$. There is a constructible (or highly uniform) 3CNF ψ over the input x and the tableau y, s.t. ψ can only be satisfied when $x \in \mathcal{L}$ (see Definition 3.12 and Proposition 3.13). In the protocol, $\mathcal{V}_{\mathsf{Ttime}}$ gets some (potentially corrupted) low-degree extension $\tilde{y}$. The prover and verifier use a Sumcheck protocol to verify that indeed $\tilde{y}$ satisfies every clause of ψ. Completeness follows by design. Unambiguity follows because:

- There is at most a single unique tableau that satisfies ψ (if $x \notin \mathcal{L}$ then there is no satisfying tableau). If $\mathcal{P}_{\mathsf{Ttime}}$ sends $\tilde{y} \neq \hat{y}$, then the tableau encoded in $\tilde{y}$ does not satisfy ψ, and this will be detected using the Sumcheck protocol.
- The Sumcheck protocol is itself unambiguous, so if the prover deviates during that protocol this will lead to rejection w.h.p.

The communication in this protocol is large: $\hat{y}$ is of size $\mathsf{poly}(T)$, but the query complexity and verification time are small (thanks to the low complexity of the Sumcheck protocol). The full protocol $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ is in Fig. 5. We then prove its (prescribed) completeness, unambiguity, and the various complexity measure bounds.

Reducing verification of $\mathcal{L}$ to a Sumcheck. We want to show soundness against encoded provers, which are restricted to send low-degree extensions in all of their messages. Once the prover sends the LDE $\tilde{y}$, we want to verify that the tableau y' encoded in $\tilde{y}$ indeed satisfies the formula ψ. That is, that $(x \circ y')$ satisfies every clause in ψ.

Interactive Proof $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ for $\mathcal{L}$

$\mathcal{P}_{\mathsf{Ttime}}$ has input: $x \in \{0,1\}^n$, $\mathcal{V}_{\mathsf{Ttime}}$ has query access to x's LDE $\hat{x} : \mathbb{F}^{m_{\mathsf{input}}} \to \mathbb{F}$.

$\mathcal{P}_{\mathsf{Ttime}}$ and $\mathcal{V}_{\mathsf{Ttime}}$ take $\psi : \{0,1\}^{n+O(T \cdot S)} \to \{0,1\}$ to be the constructible $\mathsf{3CNF}$ that verifies the T-time computation of $\mathcal{L}$ (See Proposition 3.13). Define:

- $y \in \{0,1\}^{O(T \cdot S)}$ is the unique witness that satisfies ψ, as per Proposition 3.13. Take $m_T = \log_{|\mathbb{H}|}(|y|)$, and consider y's LDE, $\hat{y} : \mathbb{F}^{m_T} \to \mathbb{F}$. $\hat{y}$ is computable in time $\mathsf{poly}(T)$.
- Take $m' = \log_{|\mathbb{H}|}(|x| + |y|)$. The polynomials $I : \mathbb{H}^{m'} \to \{0,1\}$, $K : \mathbb{H}^{m'} \to \mathbb{H}^{\max(m_{\mathsf{input}}, m_T)}$ both take as input an index $i \in [|x| + |y|]$ into ψ's input. If $i \leq n$ (i.e. i is an index into x), then I outputs 1 and K outputs i. If $i > n$, i.e. I is an index into the witness, then I outputs 0 and K outputs $(i - n)$.

  We consider I's and K's extensions over $\mathbb{F}$: $\hat{I} : \mathbb{F}^{m'} \to \mathbb{F}$ and $\hat{K} : \mathbb{F}^{m'} \to \mathbb{F}^{\max(m_{\mathsf{input}}, m_T)}$. These extensions also have individual degree $\mathsf{poly}(|\mathbb{H}|, \max(m_{\mathsf{input}}, m_T))$ and can be evaluated in time $\mathsf{poly}(|\mathbb{H}|, \max(m_{\mathsf{input}}, m_T))$.
- $\hat{C} : \mathbb{F}^{3m'+3} \to \mathbb{F}$, the circuit that specifies ψ (see Definition 3.12). $\hat{C}$ has individual degree $\mathsf{poly}(|\mathbb{H}|, m')$ and can be evaluated in time $\mathsf{poly}(|\mathbb{H}|, m')$.

With these definitions in mind, the protocol proceeds as follows:

1. If $x \notin \mathcal{L}$, $\mathcal{P}_{\mathsf{Ttime}}$ tells $\mathcal{V}_{\mathsf{Ttime}}$ to reject and $\mathcal{V}_{\mathsf{Ttime}}$ rejects immediately.

   If $x \in \mathcal{L}$, then $\mathcal{P}_{\mathsf{Ttime}}$ sends $\hat{y}$ to $\mathcal{V}_{\mathsf{Ttime}}$ in its entirety.
2. $\mathcal{V}_{\mathsf{Ttime}}$ receives $\tilde{y}$ (which is allegedly $\hat{y}$). This defines a polynomial $\tilde{P} : \mathbb{F}^{3m'+3} \to \mathbb{F}$:

   $$\tilde{P}(i_1, i_2, i_3, b_1, b_2, b_3) = \prod_{j \in \{1,2,3\}} \left((\hat{I}(i_j) \cdot \hat{x}(\hat{K}(i_j))) + ((1 - \hat{I}(i_j)) \cdot \tilde{y}(\hat{K}(i_j))) - b_j \right). \quad (8)$$

   $\mathcal{V}_{\mathsf{Ttime}}$ chooses at random $z^* \in \mathbb{F}^{3m'}$ and sends z^* to $\mathcal{P}_{\mathsf{Ttime}}$.
3. $\mathcal{P}_{\mathsf{Ttime}}$ and $\mathcal{V}_{\mathsf{Ttime}}$ run a ρ-round Sumcheck protocol $(\mathcal{P}_{\mathsf{sumchk}}, \mathcal{V}_{\mathsf{sumchk}})$ on the polynomial $Q : \mathbb{F}^{3m'+3} \to \mathbb{F}$:

   $$Q(z) \stackrel{\text{def}}{=} \tau(z, z^*) \cdot \hat{C}(z) \cdot \tilde{P}(z). \quad (9)$$

   If $\mathcal{V}_{\mathsf{sumchk}}$ rejects, then $\mathcal{V}_{\mathsf{Ttime}}$ rejects. Otherwise, $\mathcal{V}_{\mathsf{sumchk}}$ outputs $z = (i_1, i_2, i_3, b_1, b_2, b_3) \in \mathbb{F}^{3m_T+3}$ and $\nu \in \mathbb{F}$.
4. $\mathcal{V}_{\mathsf{Ttime}}$ accepts if and only if:

   $$\nu = \tau(z, z^*) \cdot \hat{C}(z) \cdot \prod_{j \in \{1,2,3\}} \left((\hat{I}(i_j) \cdot \hat{x}(\hat{K}(i_j))) + ((1 - \hat{I}(i_j)) \cdot \tilde{y}(\hat{K}(i_j))) - b_j \right). \quad (10)$$

Figure 5: Interactive Proof $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ for $\mathcal{L}$

Taking $\hat{C}$ to be the circuit that specifies ψ (see Definition 3.12), we have that (x, y') satisfies ψ if and only if for every $i_1, i_2, i_3 \in \mathbb{H}^{m'}$ and $b_1, b_2, b_3 \in \mathbb{H}$:

$$\hat{C}(i_1, i_2, i_3, b_1, b_2, b_3) \cdot \big((x \circ y')_{i_1} - b_1\big) \cdot \big((x \circ y')_{i_2} - b_2\big) \cdot \big((x \circ y')_{i_3} - b_3\big) = 0.$$

To extend this condition to the extension field $\mathbb{F}$, we define the "indexing polynomials" I and K (see Fig. 5). These are used to convert indices into ψ's input $(x \circ y)$, into indices into the input x or the witness y' (which is helpful because these two strings are encoded separately). For an index $i \in \mathbb{H}^{m'}$, the polynomial $I(i)$ tells us whether i is an input variable or a tableau variable. The polynomial K translates the index $i \in [|x| + |y|]$ to an index in $[|x|]$ into the input x (when $I(i) = 1$ and i indeed points to an input variable), or to an index in $[|y|]$ into the tableau y (when $I(i) = 0$ and i indeed points to a witness variable).

Using these indexing polynomials, taking the polynomial $P : \mathbb{F}^{3m'+3} \to \mathbb{F}$, as defined in Eq. (8) of Fig. 5, we get that (x, y') satisfy ψ if and only if:

$$\forall x \in \mathbb{H}^{3m'+3} : \ \hat{C}(z) \cdot \tilde{P}(z) = 0.$$

Thus, we have reduced checking whether (x, y') satisfies ψ (i.e. $x \in \mathcal{L}$), to checking that a low degree polynomial

$$R(z) = \hat{C}(z) \cdot \tilde{P}(z)$$

is identically zero over the subcube $\mathbb{H}^{3m'+3}$ of its domain.

Intuitively, to check this final condition, we consider the (unique) low degree extension $\widehat{R}$ of R with respect to $\mathbb{F}$, $\mathbb{H}$ and $(3m' + 3)$.[21] That is, $\widehat{R}$ is the unique polynomial of individual degree $(|\mathbb{H}| - 1)$ that agrees with R on $\mathbb{H}^{3m'+3}$. Observe that by the Schwartz-Zippel Lemma (Lemma 3.1), if R is identically zero on $\mathbb{H}^{3m'+3}$, then $\widehat{R}$ is identically zero on $\mathbb{F}^{3m'+3}$, whereas if R is *not* identically zero on $\mathbb{H}^{3m'+3}$, then $\widehat{R}$ is non-zero on most points in $\mathbb{F}^{3m'+3}$. Thus, to check whether $\psi(x, w') = 1$, the verifier $\mathcal{V}_{\mathsf{Ttime}}$ first selects a random point $z^* \in \mathbb{F}^{3m'+3}$ and sends z^* to the prover. Then, the $\mathcal{P}_{\mathsf{Ttime}}$ and $\mathcal{V}_{\mathsf{Ttime}}$ run a sub-protocol to check whether $\widehat{R}(z^*) = 0$. Since $\widehat{R}$ is a linear combination of R's evaluations on the subcube $\mathbb{H}^{3m'+3}$, we can check whether $\widehat{R}(z^*) = 0$ using the Sumcheck protocol.

Prescribed Completeness. If $x \in \mathcal{L}$ then $\psi(x, y) = 1$, and so $\forall z \in \mathbb{H}^{3m'+3}, R(z) = 0$. Recall that $\widehat{R} : \mathbb{F}^{3m'+3}$ is the *unique* polynomial of individual degree $|\mathbb{H}| - 1$ that agrees with R on $\mathbb{H}^{3m'+3}$. Since the all-zeros polynomial is such a polynomial, it must be the case that $\forall z^* \in \mathbb{F}^{3m'+3}$, $\widehat{R}(z^*) = 0$.

Hence, $\mathcal{V}_{\mathsf{Ttime}}$ and $\mathcal{P}_{\mathsf{Ttime}}$ run $(\mathcal{P}_{\mathsf{sumchk}}, \mathcal{V}_{\mathsf{sumchk}})$ on a polynomial $Q : \mathbb{F}^{3m'+3} \to \mathbb{F}$ such that

$$\sum_{z \in \mathbb{H}^{3m'+3}} Q(z) = \sum_{z \in \mathbb{H}^{3m'+3}} \tau(z, z^*) \cdot \hat{C}(z) \cdot \tilde{P}(z) = \sum_{z \in \mathbb{H}^{3m'+3}} \tau(z, z^*) \cdot R(z) \stackrel{\text{def}}{=} \widehat{R}(z^*) = 0.$$

By the completeness of the sumcheck protocol, $\mathcal{V}_{\mathsf{sumchk}}$ does not reject, and it outputs a point

[21] In Section 3.2 we only defined the low degree extension for functions over $\mathbb{H}^m$. We extend this definition to functions over $\mathbb{F}^m$ by restricting the domain to $\mathbb{H}^m$ and taking the low degree extension of the restricted function.

$z = (i_1, i_2, i_3, b_1, b_2, b_3) \in \mathbb{F}^{3m'+3}$ and a value $\nu \in \mathbb{F}$ such that $Q(z) = \nu$. Hence,

$$\begin{aligned}\nu &= Q(z) \\ &= \tau(z, z^*) \cdot \hat{C}(z) \cdot \tilde{P}(z) \\ &= \tau(z, z^*) \cdot \widehat{C}(z) \cdot \prod_{j \in \{1,2,3\}} \left((\hat{I}(i_j) \cdot \hat{x}(\hat{K}(i_j))) + ((1 - \hat{I}(i_j)) \cdot \tilde{y}(\hat{K}(i_j))) - b_j \right)\end{aligned}$$

By completeness of the Sumcheck protocol, $\mathcal{V}_{\mathsf{Ttime}}$'s check in Step 4 will be successful, and it will accept.

Unambiguity. To prove unambiguity, we consider the possible deviations that a cheating (encoded) prover $\widetilde{\mathcal{P}}$ can make. In the first case, $\widetilde{\mathcal{P}}$ deviates in Step 1, and sends an encoding $\tilde{y}$ of a witness y' where $\psi(x, y') = 0$, i.e. $(x \circ y')$ do not satisfy some clause $\tilde{z} \in \mathbb{H}^{3m'+3}$ of ψ. This could be the case either because $x \notin \mathcal{L}$, and so there is no satisfying assignment, or because even though $x \in L$, $\widetilde{\mathcal{P}}$ chooses to send $\tilde{y} \neq \hat{y}$. We emphasize that we only consider encoded cheating provers, so it is *always* the case that $\tilde{y}$ is an LDE. In this case:

$$\exists \tilde{z} \in \mathbb{H}^{3m'+3}, \text{ s.t. } R(\tilde{z}) = \hat{C}(\tilde{z}) \cdot \tilde{P}(\tilde{z}) \neq 0$$

Taking $\widehat{R}$ to be the LDE of R, by the Schwartz-Zippel Lemma (Lemma 3.1), with all but $\frac{\mathsf{poly}(|\mathbb{H}|, m')}{|\mathbb{F}|}$ probability over the choice of z^*, we have that $\widehat{R}(z^*) \neq 0$. By unambiguity of the Sumcheck protocol (Lemma 7.1), with all but $\frac{\mathsf{poly}(|\mathbb{H}|, m')}{|\mathbb{F}|}$ probability over the verifier's coin tosses in subsequent rounds, either $\mathcal{V}_{\mathsf{sumchk}}$ rejects, or it outputs $z \in \mathbb{F}^{3m'+3}$ and $\nu \in \mathbb{F}$ s.t.

$$\begin{aligned}\nu &\neq Q(z) \\ &= \tau(z, z^*) \cdot \hat{C}(z) \cdot \tilde{P}(z)) \\ &= \tau(z, z^*) \cdot \widehat{C}(z) \cdot \prod_{j \in \{1,2,3\}} \left((\hat{I}(i_j) \cdot \hat{x}(\hat{K}(i_j))) + ((1 - \hat{I}(i_j)) \cdot \tilde{y}(\hat{K}(i_j))) - b_j \right),\end{aligned}$$

and $\mathcal{V}_{\mathsf{Ttime}}$ rejects in Step 4.

For proving unambiguity, the second case is when $\widetilde{\mathcal{P}}$ does *not* deviate in its first message. That is, we have that $x \in \mathcal{L}$ and $\tilde{y} = \hat{y}$ is the LDE of w s.t. $\psi(x, w) = 1$. In this case, $\widetilde{\mathcal{P}}$ deviates within the sumcheck protocol and, by that protocol's unambiguity (Lemma 7.1), with all but $\frac{\mathsf{poly}(|\mathbb{H}|, m')}{|\mathbb{F}|}$ probability over its coin tosses, either $\mathcal{V}_{\mathsf{sumchk}}$ rejects or it outputs $z \in \mathbb{F}^{3m'+3}$ and $\nu \in \mathbb{F}$ s.t. $\nu \neq Q(z)$. Similarly to the above, this leads $\mathcal{V}_{\mathsf{Ttime}}$ to reject in Step 4.

We conclude that if $\widetilde{\mathcal{P}}$ deviates from the prescribed strategy at any point, then $\mathcal{V}_{\mathsf{Ttime}}$ rejects with all but $\frac{\mathsf{poly}(|\mathbb{H}|, m')}{|\mathbb{F}|}$ probability over its coin tosses.

Complexity measures. Recall that by the conditions of Lemma 7.3, we have that $|\mathbb{F}| = \mathsf{poly}(|\mathbb{H}|)$ and that $\log(\max(n, T)) \leq |\mathbb{H}| \leq T^{1/\rho}$. Thus, we have that $\rho \leq m' \leq (6|\mathbb{H}| + 3)$. By construction, the number of rounds is $\ell_{\mathsf{T}} = (\rho+1)$. In the first protocol message (Step 1), the prover sends an LDE $\hat{y}$ of size $\mathsf{poly}(|\mathbb{F}|)^{m_T} = \mathsf{poly}(T)$, which is computable in time $\mathsf{poly}((T)$. The verifier simply responds with $z^* \in \mathbb{F}^{3m'+3}$. After this round, the prover and verifier engage in a Sumcheck protocol for the polynomial Q. The polynomial Q has degree $\mathsf{poly}(|\mathbb{H}|, m') = \mathsf{poly}(|\mathbb{H}|)$ and can be evaluated in time

$\mathsf{poly}(|\mathbb{H}|, m') = \mathsf{poly}(|\mathbb{H}|)$ using 3 queries into $\tilde{w}$ and 3 queries into $\hat{x}$. See Lemma 7.1 and Remark 7.2 for the complexity measures of the Sumcheck protocol. The Sumcheck verifier $\mathcal{V}_{\mathsf{sumchk}}$ outputs (z, ν), and in its final check (Step 4), $\mathcal{V}_{Ttime}$ verifies Equation (10), which requires evaluating $Q(z)$ in $\mathsf{poly}(|\mathbb{H}|)$ time and 6 queries. In total we have:

- $q_{\mathsf{T}} = O(1) + q_{\mathsf{S}} = (\rho \cdot \mathsf{poly}(T)^{1/\rho} \cdot \log|\mathbb{F}|) = \mathsf{poly}(T)^{1/\rho}$
- $a_{\mathsf{T}} = \max(\mathsf{poly}(T), a_{\mathsf{S}}) = \mathsf{poly}(T)$
- $b_{\mathsf{T}} = \max(((3m'+3) \cdot \log|\mathbb{F}|), b_{\mathsf{S}}) = O(|\mathbb{H}| \cdot \log|\mathbb{F}|)$
- $\mathcal{P}\mathsf{time}_{\mathsf{T}} = (\mathsf{poly}(T) + \mathcal{P}\mathsf{time}_{\mathsf{S}}) = \mathsf{poly}(T) \cdot \mathsf{poly}(|\mathbb{H}|) = \mathsf{poly}(T)$
- $\mathcal{V}\mathsf{time}_{\mathsf{T}} = (\mathsf{poly}(|\mathbb{H}|) + \mathcal{V}\mathsf{time}_{\mathsf{S}}) = (\mathsf{poly}(T)^{1/\rho} \cdot \mathsf{poly}(|\mathbb{H}|)) = \mathsf{poly}(T)^{1/\rho}$

□

7.3 Query Reduction Transformation: Proof of Lemma 7.4

We now present the query-reduction transformation for PCIPs. We state the lemma, using a parameter ρ that controls the number of rounds (rather than the parameter σ that controls the query and verifier's time in Lemma 8.2 which appears below). The formal statement is below, and the proof follows.

Lemma 7.4 (See also Lemma 8.2)**.** *Take $\mathbb{H}$ and $\mathbb{F}$ to be constructible field ensembles where $|\mathbb{F}| \leq \mathsf{poly}(|\mathbb{H}|)$. Let $\rho = \rho(n)$ be a round parameter and $g = g(n) \geq 1$ be a parameter.*

Let $(\mathcal{P}, \mathcal{V})$ be an ϵ-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP for a language $\mathcal{L}$ w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, where $\log(\max(n, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})) \leq |\mathbb{H}| \leq \min(q, \mathcal{V}\mathsf{time})^{1/\rho}$.

Then the protocol $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ is an ϵ_{Q}-unambiguous $(q_{\mathsf{Q}}, \ell_{\mathsf{Q}}, c_{\mathsf{Q}}, \mathcal{P}\mathsf{time}_{\mathsf{Q}}, \mathcal{V}\mathsf{time}_{\mathsf{Q}})$-PCIP for the language $\mathcal{L}$ w.r.t. $(g_{\mathsf{Q}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries. Where:

- $q_{\mathsf{Q}} = \mathsf{poly}(\mathcal{V}\mathsf{time})^{1/\rho} + O(\ell \cdot g \cdot \log|\mathbb{F}|)$.
- $\ell_{\mathsf{Q}} = \ell + 2\rho + 1$.
- $a_{\mathsf{Q}} = \max(a, \mathsf{poly}(\ell, b, \mathcal{V}\mathsf{time}, |\mathbb{H}|))$.
- $b_{\mathsf{Q}} = \max(b, O(|\mathbb{H}| \cdot \log|\mathbb{F}|))$.
- $\mathcal{P}\mathsf{time}_{\mathsf{Q}} = \mathcal{P}\mathsf{time} + \mathsf{poly}(q, \ell, b, \mathcal{V}\mathsf{time})$.
- $\mathcal{V}\mathsf{time}_{\mathsf{Q}} = \mathsf{poly}(\mathcal{V}\mathsf{time})^{1/\rho} + (\mathsf{poly}(\ell, b, g, |\mathbb{H}|))$.
- $\epsilon_{\mathsf{Q}} = \epsilon + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}$.
- $g_{\mathsf{Q}} = g$.

Proof. The goal in this transformation is reducing the query complexity and verification time of $(\mathcal{P}, \mathcal{V})$, with only a moderate increase in the other parameters. The main idea is for $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Ttime}})$ to run the communication phase of the PCIP $(\mathcal{P}, \mathcal{V})$ (and *only* the communication phase). This generates a transcript (α, β) for the protocol $(\mathcal{P}, \mathcal{V})$. After this transcript is generated, $\mathcal{V}_{\mathsf{Qreduce}}$

does not explicitly make any of $\mathcal{V}$'s queries, nor does it run its decision phase. Instead, P_{Qreduce} proves to $\mathcal{V}_{\mathsf{Qreduce}}$ that $\mathcal{V}$ would have accepted the transcript.

To do so, $\mathcal{P}_{\mathsf{Qreduce}}$ sends low-degree extensions of $\mathcal{V}$'s query addresses (into the input and the transcript), and answers to those queries (each of these vectors are encoded separately). $\mathcal{P}_{\mathsf{Qreduce}}$ claims that these are encodings of the correct view v of $\mathcal{V}$ in the protocol $(\mathcal{P}, \mathcal{V})$. $\mathcal{V}_{\mathsf{Qreduce}}$ receives some LDEs of a view v', which may or may not equal the real view in the actual transcript, which has not yet been queried. $\mathcal{P}_{\mathsf{Qreduce}}$ now proves to $\mathcal{V}_{\mathsf{Qreduce}}$ that:

- The view in v' is a view that would make $\mathcal{V}$ accept in its decision phase (and that the queries specified are indeed the queries that $\mathcal{V}$ would have made). This is done using the PCIP $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ for verifying T-time computations of Lemma 7.3. Once v' is fixed by the LDEs sent in in the previous rounds, verifying $\mathcal{V}$'s decision can be done by a Turing Machine running in time $\mathcal{V}\mathsf{time}$. Thus, the query complexity and verification time in this execution of $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ will both be low. A slight subtlety is that to run the PCIP $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$, the verifier needs access to an LDE of v' in its entirety. This can be simulated from the LDEs of the query addresses and values, and an LDE of the random coins β chosen by $\mathcal{V}$ (see Proposition 3.8). The verifier $\mathcal{V}_{\mathsf{Qreduce}}$ can evaluate the LDE $\hat{\beta}$ of β in $\mathsf{poly}(|\beta|)$ time.

- It remains to verify that the view v' is consistent with the the actual input x and transcript α from the execution of $(\mathcal{P}, \mathcal{V})$ (these have not yet been queried!). Towards this, observe that $\mathcal{V}_{\mathsf{Qreduce}}$ has access to low-degree extensions of x. It has access to the low-degree extension of α, or rather to low-degree extensions of each message in α, but this can be extended to an LDE $\hat{\alpha}$ of α in its entirety (see Proposition 3.8).

  Using these low-degree extensions, $\mathcal{V}_{\mathsf{Qreduce}}$ can use Sumcheck protocols (Lemma 7.1) to verify consistency of v' with the input and the transcript. Since we already know that the view v' would make $\mathcal{V}$ accept, we conclude that $\mathcal{V}$ would have accepted in its interaction with $\mathcal{P}$. To enable the Sumcheck consistency-test, we use the LDEs of the query and addresses and query answers, as sent by the $\mathcal{P}_{\mathsf{Qreduce}}$.

  We note that the queries $\mathcal{V}_{\mathsf{Qreduce}}$ makes here are *adaptive*: e.g., to check consistency with the transcript, it first examines the LDE of query addresses, reads some values, and then queries the transcript α itself in an address that depends on the values read. Nonetheless, when interacting with the prescribed prover, the addresses in the correct LDEs are determined solely by $\mathcal{V}$'s randomness β (because $(\mathcal{P}, \mathcal{V})$ makes input-oblivious queries, see Definition 4.7). Thus, when interacting with the prescribed prover $\mathcal{P}_{\mathsf{Qreduce}}$, the addresses of queries made by the verifier $\mathcal{V}_{\mathsf{Qreduce}}$ are independent of the input, and the protocol $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ also makes input-oblivious queries.

The communication in this protocol is large: sending the view $\hat{v}$ and running the protocol $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ require $\mathsf{poly}(\mathcal{V}\mathsf{time})$ communication. On the other hand, the verifier's query complexity and computation time are greatly reduced. The full protocol $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ is in Fig. 6. After describing the protocol, we prove its (prescribed) completeness, unambiguity, and the various complexity measure bounds.

Reducing checking $\tilde{v}$'s consistency to a Sumcheck. $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ use the Sumcheck protocol to verify that the view v' encoded in the message $\tilde{v}$ sent by the untrusted prover is consistent with the input x, and the message transcript α from Step 1. This is done in Step 4.

Query-Reduced PCIP $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$

$\mathcal{P}_{\mathsf{Qreduce}}$ has input: $x \in \{0,1\}^n$, $\mathcal{V}_{\mathsf{Qreduce}}$ has query access to x's LDE $\hat{x} : \mathbb{F}^{m_{\mathsf{input}}} \to \mathbb{F}$.

1. $\mathcal{P}_{\mathsf{Qreduce}}$ and $\mathcal{V}_{\mathsf{Qreduce}}$ run the communication phase of the PCIP $(\mathcal{P}, \mathcal{V})$.
   This gives a transcript $\alpha \in \{0,1\}^{\ell \cdot a}$ of messages sent by $\mathcal{P}$, and random coins $\beta \in \{0,1\}^{\ell \cdot b}$ sent by $\mathcal{V}$. Let $Q_x \in [|x|]^{q_x}$ and $Q_\alpha \in [|\alpha|]^{q_\alpha}$ be $\mathcal{V}$'s queries into the input and transcript.
   Take $m_\alpha, m_\beta, m_{q_x}, m_{q_\alpha}$ to be logarithms to the base $|\mathbb{H}|$ of $|\alpha|, |\beta|, |q_x|, |q_a|$. Define:
   - $\widehat{\alpha} : \mathbb{F}^{m_\alpha} \to \mathbb{F}$ is α's LDE, which can be evaluated using $(g \cdot \ell)$ queries to messages α in time $\mathsf{poly}(|\mathbb{H}|)$ (recall each message in α is already encoded, see Proposition 3.8).
   - $\widehat{\beta} : \mathbb{F}^{m_\beta} \to \mathbb{F}$ is the LDEs of β, which can be evaluated in time $\mathsf{poly}(|\beta|)$.
   - $\widehat{Q_x} : \mathbb{F}^{m_{q_x}} \to \mathbb{F}^{m_{\mathsf{input}}}$ is the LDE of the function that maps an index $j \in [q_x]$ to the address of the j-th input-query within the input x
   - $\widehat{Q_\alpha} : \mathbb{F}^{m_{q_\alpha}} \to \mathbb{F}^{m_\alpha}$ is the LDE of the function that maps an index $j \in [q_a]$ to the address of the j-th transcript-query within the transcript α.
   - $\widehat{A_x} : \mathbb{F}^{m_{q_x}} \to \mathbb{F}^{m_{\mathsf{input}}}$ is the LDE of the function that maps an index $j \in [q_x]$ to the *value* read by the j-th input-query.
   - $\widehat{A_\alpha} : \mathbb{F}^{m_{q_\alpha}} \to \mathbb{F}^{m_\alpha}$ is the LDE of the function that maps an index $j \in [q_a]$ to the *value* read by the j-th transcript-query.
2. $\mathcal{P}_{\mathsf{Qreduce}}$ sends $\hat{v} = (\widehat{Q_x}, \widehat{Q_\alpha}, \widehat{A_x}, \widehat{A_\alpha})$. $\mathcal{V}_{\mathsf{Qreduce}}$ receives some $\tilde{v} = (\widetilde{Q_x}, \widetilde{Q_\alpha}, \widetilde{A_x}, \widetilde{A_\alpha})$.
3. $\mathcal{P}_{\mathsf{Qreduce}}$ and $\mathcal{V}_{\mathsf{Qreduce}}$ run the protocol $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ to verify that $\mathcal{V}$'s decision phase would succeed on the view v' specified by $(\widehat{\beta}, \widetilde{Q_x}, \widetilde{Q_\alpha}, \widetilde{A_x}, \widetilde{A_\alpha})$. If $\mathcal{V}_{\mathsf{Ttime}}$ rejects, then $\mathcal{V}_{\mathsf{Qreduce}}$ rejects immediately. Note that $\mathcal{V}_{\mathsf{Qreduce}}$ requires access to an LDE of v' in its entirety, but this can be simulated using the five LDEs in $\tilde{v}$ (see Proposition 3.8).
4. Verifying that $\tilde{v}$ is consistent with (β, x, α) obtained in Step 1:
   (a) $\mathcal{V}_{\mathsf{Qreduce}}$ sends to $\mathcal{P}_{\mathsf{Qreduce}}$ uniformly random $j_x^* \in \mathbb{F}^{m_{q_x}}$, $j_\alpha^* \in \mathbb{F}^{m_{q_\alpha}}$.
   (b) $\mathcal{P}_{\mathsf{Qreduce}}$ and $\mathcal{V}_{\mathsf{Qreduce}}$ run two instances of the Sumcheck protocol $(\mathcal{P}_{\mathsf{sumchk}}, \mathcal{V}_{\mathsf{sumchk}})$ in parallel, on the polynomials:
   $$P_x : \mathbb{F}^{m_{q_x}} \to \mathbb{F}, \text{ where } P_x(j) = \tau(j, j_x^*) \cdot \left(\widetilde{A_x}(j) - \widehat{x}(\widetilde{Q_x}(j))\right)$$
   $$P_\alpha : \mathbb{F}^{m_{q_\alpha}} \to \mathbb{F}, \text{ where } P_\alpha(j_\alpha) = \tau(j, j_\alpha^*) \cdot \left(\widetilde{A_\alpha}(j) - \widehat{\alpha}(\widetilde{Q_\alpha}(j))\right)$$
   (c) If $\mathcal{V}_{\mathsf{sumchk}}$ rejects, then $\mathcal{V}_{\mathsf{Qreduce}}$ immediately rejects. Otherwise, $\mathcal{V}_{\mathsf{sumchk}}$ outputs (j_x, ν_x), (j_α, ν_α). $\mathcal{V}_{\mathsf{Qreduce}}$ accepts if and only if $\nu_x = P_x(j_x)$, and $\nu_\alpha = P_\alpha(j_\alpha)$.

Figure 6: Query Reduction for PCIP $(\mathcal{P}, \mathcal{V})$

Take $v' = (Q_x', Q_\alpha', A_x', A_\alpha')$ to be the message encoded in $\tilde{v} = (\widetilde{Q_x}, \widetilde{Q_\alpha}, \widetilde{A_x}, \widetilde{A_\alpha})$ (recall that since we only show unambiguity against encoded provers, we can assume that every message sent, an in particular $\tilde{v}$, is indeed composed of LDEs). $\mathcal{V}_{\mathsf{Qreduce}}$ wants to verify that $v' = v = (Q_x, Q_\alpha, A_x, A_\alpha)$. The PCIP $(\mathcal{P}_{\mathsf{Ttime}, \mathcal{V}_{\mathsf{Ttime}}})$ verifies that Q_x, Q_α are the correct queries that $\mathcal{V}$ would make using randomness β. It remains to check that the actual *answers* in A_x', A_α' are equal the (corresponding) values in $\hat{x}, \hat{\alpha}$ at the addresses specified by the queries in Q_x', Q_α'. for x, and α are consistent with the transcript and input. To perform this consistency check, $\mathcal{V}_{\mathsf{Qreduce}}$ uses low-degree extensions of α, of β and of the input x. These are computed as follows:

- We already assume that $\mathcal{V}_{\mathsf{Qreduce}}$ has query access to an LDE $\widehat{x}$ of its input.
- The transcript of prover messages α is "almost" an LDE: each of the $(\ell \cdot g)$ messages in the transcript is itself an LDE (because we only consider encoded provers). $\mathcal{V}_{\mathsf{Qreduce}}$ can evaluate any coordinate in the LDE $\widehat{\alpha}$ of the entire transcript (i.e. the LDE obtained by decoding each plaintext message in α and then re-encoding the entire transcript of plaintext messages) by making $(\ell \cdot g \cdot \log |\mathbb{F}|)$ queries to α and using time $\mathsf{poly}(|\mathbb{H}|, \ell, g)$.
- $\mathcal{V}_{\mathsf{Qreduce}}$ computes the LDE $\widehat{\beta}$ of β "from scratch" in time $((|\beta|) \cdot \mathsf{polylog}|\mathbb{F}|) = (\ell \cdot b \cdot \mathsf{polylog}|\mathbb{F}|)$.

These LDEs all have individual degree $(|\mathbb{H}| - 1)$.

The consistency check boils down to the following conditions:

- A_x' is consistent with x, i.e. for every $j \in [|q_x|]$, $A_x'(j) = x(Q_x'(j))$. Equivalently:

$$\forall j \in \mathbb{H}^{m_{q_x}} : \; R_x(j) = \left(\widetilde{A_x}(j) - \widehat{x}(\widetilde{Q_x}(j))\right) = 0 \tag{11}$$

- A_α' is consistent with α, i.e. for every $j \in [|q_\alpha|]$, $A_\alpha'(j) = \alpha(Q_\alpha'(j))$. Equivalently:

$$\forall j \in \mathbb{H}^{m_{q_\alpha}} : \; R_\alpha(j) = \left(\widetilde{A_\alpha}(j) - \widehat{\alpha}(\widetilde{Q_\alpha}(j))\right) = 0 \tag{12}$$

We want to verify that the polynomials R_x, R_α vanish over their respective $\mathbb{H}$-subcubes. This happens if and only if their (unique) low-degree extensions $\widehat{R}_x, \widehat{R}_\alpha$ are identically 0. By the Schwartz-Zippel Lemma, we know that if any of these (low-degree) polynomials is non-zero, then it is non-zero almost everywhere. Thus, $\mathcal{V}_{\mathsf{Qreduce}}$ chooses uniformly random $j_x^* \in \mathbb{F}^{m_{q_x}}$, $j_\alpha^* \in \mathbb{F}^{m_{q_\alpha}}$, and verifies that $\widehat{R}_x(j_x^*) = 0$ and that $\widehat{R}_\alpha(j_\alpha^*) = 0$.

These final zero-tests are performed using the Sumcheck protocol. Recall the definitions of P_x, P_α defined in Step 4b of Figure 6. We have that:

$$\begin{aligned} \widehat{R}_x(j_x^*) &= \sum_{j \in \mathbb{H}^{m_{q_x}}} P_x(j) \\ \widehat{R}_\beta(j_\alpha^*) &= \sum_{j \in \mathbb{H}^{m_{q_\alpha}}} P_\alpha(j) \end{aligned}$$

Thus, $\mathcal{V}_{\mathsf{Qreduce}}$ can use two Sumcheck protocols (run in parallel) to check that the polynomials $\widehat{R}_x, \widehat{R}_\alpha$ vanish on the randomly chosen coordinates j_x^*, j_α^* (respectively). $\mathcal{V}_{\mathsf{Qreduce}}$ accepts if and only if in all three of these Sumchecks the verifier $\mathcal{V}_{\mathsf{sumchk}}$ accepts and outputs a correct claim.

Prescribed Completeness. Completeness follows from the above. For $x \in \mathcal{L}$, suppose that $\mathcal{P}_{\mathsf{Qreduce}}$ follows the prescribed strategy $\mathcal{P}$ in Step 1, and sends the correct LDEs $\widehat{v}$ in Step 2. Since $\mathcal{P}_{\mathsf{Qreduce}}$ sent the correct LDE, the verifier $\mathcal{V}$ would indeed make the queries specified in v. By prescribed completeness of $(\mathcal{P}, \mathcal{V})$, $\mathcal{V}$ would accept given the view v. If $\mathcal{P}$ follows the prescribed strategy in Step 3, then by prescribed completeness of the protocol $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$, $\mathcal{V}_{\mathsf{Ttime}}$ will accept. Also, Equations (11), and (12) hold. Thus, it will indeed be the case that $\widehat{R}_x(j^*_x) = 0$, and that $\widehat{R}_\alpha(j^*_\alpha) = 0$. By prescribed completeness of the Sumcheck protocol, the sumcheck verifier will accept in all three executions, and output correct claims. The check in Step 4c will succeed, and $\mathcal{V}_{\mathsf{Qreduce}}$ will accept.

We note here again that while the verifier $\mathcal{V}_{\mathsf{Qreduce}}$ makes adaptive queries, if $(\mathcal{P}, \mathcal{V})$ makes input-oblivious queries (see Definition 4.7), then so does $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$

Unambiguity. We consider the case that $x \in \mathcal{L}$ (the case that $x \notin \mathcal{L}$ follows similarly). Let $\widetilde{\mathcal{P}}$ be a deviating prover. Consider the possible deviations:

- Deviating in Step 1. If $\widetilde{\mathcal{P}}$ deviates in the execution of $(\mathcal{P}, \mathcal{V})$, then by unambiguity of that protocol we know that with all but ϵ probability over the choice of randomness β for $\mathcal{V}$, the verifier $\mathcal{V}$ would *reject* given the (correct) view v. Suppose that $\widetilde{\mathcal{P}}$ sends $\tilde{v} = \widehat{v}$ (where $\widehat{v}$ includes LDEs of the correct view v), or that $\widetilde{\mathcal{P}}$ sends $\tilde{v}$ encoding a corrupted view v' where the query sets Q'_x, Q'_α are not consistent with the choice of randomness β and the answers A'_x, A'_α (recall that the verifier $\mathcal{V}$ may be adaptive in its queries). In either of these cases, by unambiguity of $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ (Lemma 7.3), the verifier $\mathcal{V}_{\mathsf{Ttime}}$ will reject in Step 3 with all but ϵ_{T} probability.

  The remaining case is that $\widetilde{\mathcal{P}}$ sends $\tilde{v} \neq \widehat{v}$ where $\tilde{v}$ encodes a view v' containing the correct query sets Q_x, Q_α. In this case, we know that v' is not consistent with either x or α. Thus, one of the Equations (11) and (12) is not satisfied. By the Schwartz-Zippel Lemma, with all but $\frac{\mathsf{poly}(|\mathbb{H}|, m')}{|\mathbb{F}|}$ probability over the choice of j^*_x, j^*_α, it will be the case that either $\widehat{R}_x(j^*_x) \neq 0$ or $\widehat{R}_\alpha(j^*_\alpha) \neq 0$. When this is the case, by unambiguity of the Sumcheck Protocol (Lemma 7.1), with all but $\frac{\mathsf{poly}(|\mathbb{H}|, m')}{|\mathbb{F}|}$ over the coins of $\mathcal{V}_{\mathsf{sumchk}}$ in the two executions, in at least one of the executions either $\mathcal{V}_{\mathsf{sumchk}}$ rejects, (and $\mathcal{V}_{\mathsf{Qreduce}}$ rejects immediately), or it outputs an incorrect claim about one of the polynomials P_x, P_α, and $\mathcal{V}_{\mathsf{Qreduce}}$ rejects.

  We conclude that if $\widetilde{\mathcal{P}}$ first deviates in Step 1, then $\mathcal{V}_{\mathsf{Qreduce}}$ rejects with all but $(\epsilon + \epsilon_{\mathsf{T}} + \frac{\mathsf{poly}(|\mathbb{H}|, m')}{|\mathbb{F}|})$ probability over its subsequent coins tosses.

- Deviating in Step 2. If $\widetilde{\mathcal{P}}$ first deviates in sending $\tilde{v}$, then we know that it followed the prescribed strategy $\mathcal{P}$ in Step 1. By prescribed completeness of $(\mathcal{P}, \mathcal{V})$, we know that $\mathcal{V}$ would accept given the (correct) view v, whose LDEs are $\widehat{v}$. Since $\widetilde{\mathcal{P}}$ deviates in Step 2, it sends LDEs $\tilde{v} \neq \widehat{v}$. Let v' be the view encoded in $\tilde{v}$. If the query sets Q'_x, Q'_α in v' are not consistent with the choice of randomness β and the given answers A'_x, A'_α, then by unambiguity of $(\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}})$ (Lemma 7.3), the verifier $\mathcal{V}_{\mathsf{Ttime}}$ will reject in Step 3 with all but ϵ_{T} probability.

  The remaining case is that $\widetilde{\mathcal{P}}$ sent $\tilde{v} \neq \widehat{v}$ where $\tilde{v}$ encodes a view v' containing the correct query sets Q_x, Q_α. In this case, we know that v' is not consistent with either x or α. Following

the same analysis as we did for the case of a deviation in Step 1, we conclude that in this case $\mathcal{V}_{\mathsf{Qreduce}}$ will reject with all but $\frac{\mathsf{poly}(|\mathbb{H}|,m')}{|\mathbb{F}|}$ probability.

We conclude that if $\widetilde{\mathcal{P}}$ first deviates in Step 2, then $\mathcal{V}_{\mathsf{Qreduce}}$ rejects with all but $(\epsilon_{\mathsf{T}}+\frac{\mathsf{poly}(|\mathbb{H}|,m')}{|\mathbb{F}|})$ probability over its subsequent coins tosses.

- Deviating in Step 3. In this case, $\widetilde{\mathcal{P}}$ follows the prescribed strategy in executing the protocol $(\mathcal{P},\mathcal{V})$ and sends LDEs $\tilde{v}=\widehat{v}$ of the correct view v. Thus, in the prescribed execution of the PCIP $(\mathcal{P}_{\mathsf{Ttime}},\mathcal{V}_{\mathsf{Ttime}})$, the verifier $\mathcal{V}_{\mathsf{Ttime}}$ should accept the view v. However, since $\widetilde{\mathcal{P}}$ deviates in $(\mathcal{P}_{\mathsf{Ttime}},\mathcal{V}_{\mathsf{Ttime}})$, by unambiguity of that protocol (Lemma 7.3), the verifier $\mathcal{V}_{\mathsf{Ttime}}$ will reject with all but ϵ_{T} probability.

  We conclude that if $\widetilde{\mathcal{P}}$ first deviates in Step 3, then $\mathcal{V}_{\mathsf{Qreduce}}$ will reject with all but ϵ_{T} probability.

- Deviating in Step 4b. In this case, $\widetilde{\mathcal{P}}$ follows the prescribed strategy in in executing the protocol $(\mathcal{P},\mathcal{V})$, sends LDEs $\tilde{v}=\widehat{v}$ of the correct view v, and follows the prescribed strategy in executing $(\mathcal{P}_{\mathsf{Ttime}},\mathcal{V}_{\mathsf{Ttime}})$ (and so $\mathcal{V}_{\mathsf{Ttime}}$ accepts). However, $\widetilde{\mathcal{P}}$ deviates in at least one of the executions of the Sumcheck protocol. By that protocol's unambiguity (Lemma 7.1, with all but $\frac{\mathsf{poly}(|\mathbb{H}|,m')}{|\mathbb{F}|}$ probability over the choice of coins for the executions of $\mathcal{V}_{\mathsf{sumchk}}$, either it rejects (and $\mathcal{V}_{\mathsf{Qreduce}}$ rejects immediately), or it outputs an incorrect claim about at least one of the polynomials P_x, P_α, and $\mathcal{V}_{\mathsf{Qreduce}}$ rejects.

  We conclude that if $\widetilde{\mathcal{P}}$ first deviates in Step 3, then $\mathcal{V}_{\mathsf{Qreduce}}$ will reject with all but $\frac{\mathsf{poly}(|\mathbb{H}|,m')}{|\mathbb{F}|}$ probability.

Unambiguity for the case that $x\notin\mathcal{L}$ follows similarly (it is only easier to argue that $\mathcal{V}_{\mathsf{Qreduce}}$ rejects, because if $\widetilde{\mathcal{P}}$ does not deviate in Step 2 then the "real" view v would make $\mathcal{V}$ reject). We conclude that $(\mathcal{P}_{\mathsf{Qreduce}},\mathcal{V}_{\mathsf{Qreduce}})$ is an ϵ_{Q}-unambiguous PCIP, where

$$\epsilon_{\mathsf{Q}}=(\epsilon+\epsilon_{\mathsf{T}}+\frac{\mathsf{poly}(|\mathbb{H}|,m')}{|\mathbb{F}|})=(\epsilon+\frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}),$$

where we used the facts that $m'=O(|\mathbb{H}|)$ (see below) and $\epsilon_{\mathsf{T}}=\frac{|\mathbb{H}|}{|\mathbb{F}|}$ (Lemma 7.3).

Complexity measures. Recall that by the conditions of Lemma 7.3, we have that $|\mathbb{F}|=\mathsf{poly}(|\mathbb{H}|)$ and that $\log(\max(n,\ell,a,b,\mathcal{P}\mathsf{time},\mathcal{V}\mathsf{time}))\leq|\mathbb{H}|\leq(\min(q,\mathcal{V}\mathsf{time}))^{1/\rho}$. Indeed, w.l.o.g. we will even assume that $|\mathbb{H}|\leq(\min(q_x,q_\alpha,\mathcal{V}\mathsf{time}))^{1/\rho}$. Thus, we have that $\rho\leq\min(m',m_{q_x},m_{q_\alpha})\leq 8|\mathbb{H}|$.

By construction, the number of rounds is $(\ell+2\rho+1)$: ℓ rounds for running $(\mathcal{P},\mathcal{V})$, a single message from the prover containing $\tilde{v}$ (can be piggy-backed onto the next protocol), $(\rho+1)$ rounds for running $(\mathcal{P}_{\mathsf{Ttime}},\mathcal{V}_{\mathsf{Ttime}})$, a single message from the verifier containing j^*_x,j^*_α (can be piggy-backed onto the previous protocol), and two parallel executions of the Sumcheck protocol, which require ρ rounds. We note that we could even run Steps 3 and 4b in parallel (and save ρ rounds), but we run them sequentially for ease of presentation.

We consider the prover and verifier complexity in each of the protocol steps:

- In Step 1, $(\mathcal{P}_{\mathsf{Qreduce}},\mathcal{V}_{\mathsf{Qreduce}})$ run $(\mathcal{P},\mathcal{V})$, which is a $(q,\ell,a,b,\mathcal{P}\mathsf{time},\mathcal{V}\mathsf{time})$-PCIP. They only generate the transcript, without making any of $\mathcal{V}$'s queries or computation. Thus, the message length are as in $(\mathcal{P},\mathcal{V})$, as is the prover runtime, but the verifier does not pay any queries or computation.

- In Step 2, $\mathcal{P}_{\mathsf{Qreduce}}$ sends the LDEs of $\mathcal{V}$'s view v. The view v itself is of length $O(q \cdot \max(\log n, \log(\ell \cdot a)))$. The prover $\mathcal{P}_{\mathsf{Qreduce}}$ can compute v and its LDE in time $\mathsf{poly}(q, \ell, b, \mathcal{V}\mathsf{time}, |\mathbb{H}|)$, and the length of the LDE is $\mathsf{poly}(q, \ell, |\mathbb{H}|)$.

- In Step 3, ($\mathcal{P}_{\mathsf{Qreduce}}$ and $\mathcal{V}_{\mathsf{Qreduce}}$) run the ($\mathcal{P}_{\mathsf{Ttime}}, \mathcal{V}_{\mathsf{Ttime}}$) PCIP on the LDE of the view v encoded in $\widehat{v}$, and verify the results of $\mathcal{V}$'s $\mathcal{V}\mathsf{time}$-time computation. By Lemma 7.3, this requires queries $q_{\mathsf{T}} = \mathsf{poly}(\mathcal{V}\mathsf{time})^{1/\rho}$, prover message length $a_{\mathsf{T}} = \mathsf{poly}(\mathcal{V}\mathsf{time})$, verifier message length $b_{\mathsf{T}} = O(|\mathbb{H}| \cdot \log |\mathbb{F}|)$, prover time $\mathcal{P}\mathsf{time}_{\mathsf{T}} = \mathsf{poly}(\mathcal{V}\mathsf{time})$, and verifier time $\mathcal{V}\mathsf{time}_{\mathsf{T}} = \mathsf{poly}(\mathcal{V}\mathsf{time})^{1/\rho}$. Each query to the LDE of v (in its entirety) can be simulated by $\mathcal{V}_{\mathsf{Qreduce}}$ using $\mathsf{poly}(|\mathbb{H}|)$ queries to $\widehat{v}$, and a query to $\widehat{\beta}$. The entire LDE $\widehat{\beta}$ can be computed in time $\mathsf{poly}(|\beta|, |\mathbb{H}|) = \mathsf{poly}(\ell, b, |\mathbb{H}|)$.

- In Step 4a, $\mathcal{V}_{\mathsf{Qreduce}}$ sends j^*_x, j^*_α. The message length is $((m_{q_x} + m_{q_\alpha}) \cdot \log |\mathbb{F}|) = O(|\mathbb{H}| \cdot \log |\mathbb{F}|)$.

- In Step 4b $\mathcal{P}_{\mathsf{Qreduce}}$ and $\mathcal{V}_{\mathsf{Qreduce}}$ run two sumcheck protocols for the polynomials Q_x, Q_α. These polynomials have individual degree $\mathsf{poly}(|\mathbb{H}|)$ and input lengths $m_{q_x}, m_{q_\alpha} \leq |\mathbb{H}|$. Recall that the Lemma statement assumes $|\mathbb{H}| \leq \min(q_x, q_\alpha)^{1/\rho}$, and so $\rho \leq m_{q_x}, m_{q_\alpha}$.

  By Lemma 7.1 and Remark 7.2, running the sumcheck protocols requires queries $q_{\mathsf{sumchk}} = \mathsf{poly}(q)^{1/\rho}$, prover messages of length $a_{\mathsf{S}} = \mathsf{poly}(q)^{1/\rho}$, verifier messages of length $b_{\mathsf{S}} = (|\mathbb{H}| \cdot \log |\mathbb{F}|)$, Prover time $\mathcal{P}\mathsf{time}_{\mathsf{S}} = \mathsf{poly}(q)$ and Verifier time $\mathcal{V}\mathsf{time}_{\mathsf{S}} = \mathsf{poly}(q)^{1/\rho}$.

- In Step 4c, $\mathcal{V}_{\mathsf{Qreduce}}$ needs to evaluate the polynomials P_x, P_α (each at a single coordinate). These evaluations require $O(\ell \cdot g \cdot \log |\mathbb{F}|)$ queries and time $\mathsf{poly}(\ell, g, |\mathbb{H}|)$ (see the discussion above, this requires computing the LDEs $\widehat{\alpha}, K_x, K_\alpha, K_{Q_x}, K_{Q_\alpha}$).

Summing up the costs of all steps above, we get:

- $q_{\mathsf{Q}} = (q_{\mathsf{T}} + 1 + q_{\mathsf{sumchk}} + O(\ell \cdot g \cdot \log |\mathbb{F}|)) = \left(\mathsf{poly}(\mathcal{V}\mathsf{time}, q)^{1/\rho} + O(\ell \cdot g \cdot \log |\mathbb{F}|)\right).$

- $a_{\mathsf{Q}} = \max\left(a, \mathsf{poly}(q, \ell, b, |\mathbb{H}|), a_{\mathsf{T}}, a_{\mathsf{S}}\right) = \max\left(a, \mathsf{poly}(q, \ell, b, |\mathbb{H}|), \mathsf{poly}(\mathcal{V}\mathsf{time})\right).$

- $b_{\mathsf{Q}} = \max\left(b, b_{\mathsf{T}}, O(|\mathbb{H}| \cdot \log |\mathbb{F}|)\right) = \max\left(b, O(|\mathbb{H}| \cdot \log |\mathbb{F}|)\right).$

- $\mathcal{P}\mathsf{time}_{\mathsf{Q}} = (\mathcal{P}\mathsf{time} + \mathsf{poly}(q, \ell, b, \mathcal{V}\mathsf{time}) + \mathcal{P}\mathsf{time}_{\mathsf{T}} + \mathcal{P}\mathsf{time}_{\mathsf{S}}) = (\mathcal{P}\mathsf{time} + \mathsf{poly}(q, \ell, b, \mathcal{V}\mathsf{time})).$

- $$\begin{aligned}\mathcal{V}\mathsf{time}_{\mathsf{Q}} &= (\mathcal{V}\mathsf{time}_{\mathsf{T}} + \mathsf{poly}(\ell, b, |\mathbb{H}|) + \mathcal{V}\mathsf{time}_{\mathsf{S}} + \mathsf{poly}(\ell, g, |\mathbb{H}|)) \\ &= \left(\mathsf{poly}(\mathcal{V}\mathsf{time})^{1/\rho} + (b \cdot \mathsf{poly}(\ell, b, g, |\mathbb{H}|))\right).\end{aligned}$$

□

8 Interactive Proofs for Bounded-Space Computations

In this section we construct interactive proofs for bounded-space computations, proving our main results (see Section 5, in particular Theorem 7 and Corollaries 8 and 9). As described in the overview (Section 2), the construction is iterative, and uses the two transformations on unambiguous PCIPs (probabilistically checkable interactive proofs) that were shown in Section 6 and Section 7, respectively. Before proceeding to the proof, we first restate the batch verification and query reduction results of Sections 6 and 7.

Recall that if $\mathcal{L}$ is a language, we defined $\mathcal{L}^{\otimes k}$ as the language that contains all k-tuples of elements in $\mathcal{L}$ of the same length (see the beginning of Section 6 for the formal definition). The following lemma (which is essentially a re-statement of Lemma 6.1) shows that suitable PCIPs for a language $\mathcal{L}$ can be batched to produce a non-trivial PCIP for $\mathcal{L}^{\otimes k}$.

Lemma 8.1 (Batch Verification for unambiguous PCIPs w.r.t. encoded provers (c.f. Lemma 6.1))**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$ and $|\mathbb{F}| \leq \mathsf{poly}(|\mathbb{H}|)$.*

Let $(\mathcal{P}, \mathcal{V})$ be an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP for the language $\mathcal{L}$, with respect to $(g, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries. Let $k = k(n) \geq 1$, $\sigma = \sigma(n) \in (0, 1)$ and let $\lambda = \lambda(n) \geq 1$ be a security parameter, where:

- $\log(\ell^{1/\sigma} \cdot a) \leq \min\left(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\right)$.
- $a \geq \mathsf{poly}(k, q, g) \cdot (\mathsf{poly}(\lambda, |\mathbb{H}|, \ell))^{1/\sigma}$.
- $\mathcal{P}\mathsf{time} \geq \ell \cdot a \cdot \mathsf{polylog}(|\mathbb{F}|)$.

Then, there exists an ε_{A}-unambiguous $(q_{\mathsf{A}}, \ell_{\mathsf{A}}, a_{\mathsf{A}}, b_{\mathsf{A}}, \mathcal{P}\mathsf{time}_{\mathsf{A}}, \mathcal{V}\mathsf{time}_{\mathsf{A}})$-PCIP for the language $\mathcal{L}^{\otimes k}$ w.r.t. $(g_{\mathsf{A}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, with the following parameters:

- $\varepsilon_{\mathsf{A}} = (\frac{1}{\sigma} + 1) \cdot \left(\varepsilon + 3 \cdot 2^{-\lambda}\right)$.
- $q_{\mathsf{A}} = k^2 \cdot q \cdot \mathsf{poly}(g) \cdot \left(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\right)^{1/\sigma}$.
- $\ell_{\mathsf{A}} = O(\ell/\sigma)$.
- $b_{\mathsf{A}} = \max\left(b, k \cdot \mathsf{poly}(|\mathbb{H}|, \ell^{1/\sigma}, g, \lambda)\right)$.
- $a_{\mathsf{A}} = a \cdot 2k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}$.
- $\mathcal{P}\mathsf{time}_{\mathsf{A}} = 2k \cdot (1/\sigma) \cdot \ell^{1/\sigma} \cdot \mathcal{P}\mathsf{time} + \mathcal{V}\mathsf{time} \cdot q \cdot k^2 \cdot \mathsf{poly}(g) \cdot \left(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\right)^{1/\sigma}$.
- $\mathcal{V}\mathsf{time}_{\mathsf{A}} = \mathcal{V}\mathsf{time} \cdot q \cdot k^2 \cdot \mathsf{poly}(g) \cdot \left(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\right)^{1/\sigma}$.
- $g_{\mathsf{A}} = g \cdot 2k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}$.

Our second main ingredient is a "query reduction" transformation for unambiguous PCIPs (w.r.t. encoded provers) that reduces the verifier's query complexity and running time. The following is essentially a restatement of Lemma 7.4.

Lemma 8.2 (Query-reduction for unambiguous PCIPs w.r.t. encoded provers (c.f. Lemma 7.4))**.** *Take $\mathbb{H}$ and $\mathbb{F}$ to be constructible field ensembles where $|\mathbb{F}| = \mathsf{poly}(|\mathbb{H}|)$. Let $\sigma = \sigma(n) \in (0, 1)$ be a* reduction *parameter and $g = g(n) \geq 1$ a parameter.*

Let $(\mathcal{P}, \mathcal{V})$ be an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP for a language $\mathcal{L}$ w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, where $\log(\max(n, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})) \leq |\mathbb{H}| \leq \min(q, \mathcal{V}\mathsf{time})^{\sigma}$.

There exists an ε_{Q}-unambiguous $(q_{\mathsf{Q}}, \ell_{\mathsf{Q}}, a_{\mathsf{Q}}, b_{\mathsf{Q}}, \mathcal{P}\mathsf{time}_{\mathsf{Q}}, \mathcal{V}\mathsf{time}_{\mathsf{Q}})$-PCIP protocol $(\mathcal{P}_{\mathsf{Qreduce}}, \mathcal{V}_{\mathsf{Qreduce}})$ for the language $\mathcal{L}$ w.r.t. $(g_{\mathsf{Q}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries. Where:

- $q_{\mathsf{Q}} = \mathsf{poly}(\mathcal{V}\mathsf{time})^{\sigma} + O(\ell \cdot g \cdot \log|\mathbb{F}|)$.
- $\ell_{\mathsf{Q}} = \ell + O(1/\sigma)$.
- $a_{\mathsf{Q}} = \max(a, \mathsf{poly}(\ell, b, \mathcal{V}\mathsf{time}, |\mathbb{H}|))$.
- $b_{\mathsf{Q}} = \max(b, O(|\mathbb{H}| \cdot \log|\mathbb{F}|))$.
- $\mathcal{P}\mathsf{time}_{\mathsf{Q}} = \mathcal{P}\mathsf{time} + \mathsf{poly}(q, \ell, b, \mathcal{V}\mathsf{time})$.
- $\mathcal{V}\mathsf{time}_{\mathsf{Q}} = \mathsf{poly}(\mathcal{V}\mathsf{time})^{\sigma} + (\mathsf{poly}(b, \ell, g, |\mathbb{H}|))$.
- $\varepsilon_{\mathsf{Q}} = \varepsilon + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}$.
- $g_{\mathsf{Q}} = g$.

Lemmas 8.1 and 8.2 follow immediately from Lemmas 6.1 and 7.4, respectively.

8.1 Augmentation Lemma

As described in the technical overview (Section 2), our construction is based on iterative and interleaved applications of the Batch Verification Lemma (Lemma 8.1) and the Query Reduction Lemma (Lemma 8.2) above. Using these two lemmas, we show an efficient transformation from unambiguous PCIPs that verify computations of length t to unambiguous PCIPs that verify computations of length $k \cdot t$. We call this step the augmentation lemma (since it augments the length of computations).

Before presenting the lemma, we require the following definition. For a space $S = S(n)$ Turing machine $\mathcal{M}$ and a time bound $t = t(n)$, we define the language $\mathcal{L}_t^{\mathcal{M}}$ as

$$\mathcal{L}_t^{\mathcal{M}} \stackrel{\text{def}}{=} \left\{ (x,u,v) \ : \ \begin{array}{c} \text{On input } x \in \{0,1\}^n, \ \mathcal{M} \text{ moves from configuration } u \in \{0,1\}^{O(S)} \\ \text{to configuration } v \in \{0,1\}^{O(S)} \text{ in exactly } t \text{ steps} \end{array} \right\}.$$

If the machine $\mathcal{M}$ is clear from the context, then we omit it from the notation. The following lemma "augments" an unambiguous PCIP for a language $\mathcal{L}_t$ to an unambiguous PCIP for $\mathcal{L}_{k \cdot t}$.

Lemma 8.3 (Augmentation Lemma)**.** *Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$ and $|\mathbb{F}| \leq \mathsf{poly}(|\mathbb{H}|)$.*

Let $\mathcal{M}$ be a Turing machine that uses space at most $S = S(n)$. Let $t = t(n)$ be a time bound and suppose that $\mathcal{L}_t^{\mathcal{M}}$ has an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries. Let $k = k(n) \geq 1$ and $\sigma = \sigma(n) \in (0,1)$ be parameters. Assume that

1. $\log(\ell^{1/\sigma} \cdot a) \leq \min\left(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\right)$.
2. $a \geq \max\left(\mathsf{poly}(k, S), \mathsf{poly}(k, \mathcal{V}\mathsf{time}, g) \cdot (\mathsf{poly}(\lambda, |\mathbb{H}|, \ell))^{1/\sigma}\right)$.
3. $|\mathbb{H}| \geq \log\left(\mathsf{poly}\big(k, n, S, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}\big) \cdot \big(\mathsf{poly}(\lambda, |\mathbb{H}|, \ell)\big)^{1/\sigma}\right)$.
4. $\mathcal{P}\mathsf{time} \geq \ell \cdot a \cdot \mathsf{polylog}(|\mathbb{F}|)$

5. $\mathcal{V}\mathsf{time} \geq \max(q, b \cdot \ell)$.

Then, the language $\mathcal{L}^{\mathcal{M}}_{k\cdot t}$ has an $\varepsilon_{\mathsf{aug}}$-unambiguous $(q_{\mathsf{aug}}, \ell_{\mathsf{aug}}, a_{\mathsf{aug}}, b_{\mathsf{aug}}, \mathcal{P}\mathsf{time}_{\mathsf{aug}}, \mathcal{V}\mathsf{time}_{\mathsf{aug}})$-PCIP w.r.t. $(g_{\mathsf{aug}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and the following parameters:

- $q_{\mathsf{aug}} = \Big(\big(\mathcal{V}\mathsf{time} \cdot k \cdot g\big)^{\sigma} \cdot \mathsf{poly}(\ell, \lambda, |\mathbb{H}|) + O(k^{\sigma} \cdot \ell^{O(1/\sigma)} \cdot g \cdot \log(k) \cdot \lambda \cdot \mathsf{poly}(|\mathbb{H}|))\Big)$.
- $\ell_{\mathsf{aug}} = O(\ell/\sigma)$.
- $a_{\mathsf{aug}} = \big(2a \cdot k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}\big)$.
- $b_{\mathsf{aug}} = \max\big(b, k \cdot \mathsf{poly}(|\mathbb{H}|, \ell^{1/\sigma}, g, \lambda)\big)$.
- $\mathcal{P}\mathsf{time}_{\mathsf{aug}} = \Big(k\cdot t + \mathsf{poly}(k, S) + O\big(k \cdot \mathcal{P}\mathsf{time} \cdot 1/\sigma \cdot \ell^{1/\sigma}\big) + \mathsf{poly}\big(\mathcal{V}\mathsf{time}, k, g, \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{1/\sigma}\big)\Big)$.
- $\mathcal{V}\mathsf{time}_{\mathsf{aug}} = \Big((\mathcal{V}\mathsf{time} \cdot k \cdot g)^{\sigma} \cdot \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big) + \mathsf{poly}(k, b, |\mathbb{H}|, \ell^{1/\sigma}, g, \lambda)\Big)$.
- $\varepsilon_{\mathsf{aug}} = \Big((\frac{1}{\sigma} + 1) \cdot \big(\varepsilon + 3 \cdot 2^{-\lambda}\big) + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}\Big)$.
- $g_{\mathsf{aug}} = \big(g \cdot 2k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}\big)$.

Most importantly, observe that the verification time $\mathcal{V}\mathsf{time}_{\mathsf{aug}}$ and prover message length a_{aug} increase only by roughly a k^{σ} factor (rather than the trivial k factor).

Proof of Lemma 8.3. Let $\mathcal{M}$ be a Turing machine that uses space at most $S = S(n)$. For convenience we use the notation $\mathcal{L}_t \stackrel{\text{def}}{=} \mathcal{L}^{\mathcal{M}}_t$ and $\mathcal{L}_{k\cdot t} \stackrel{\text{def}}{=} \mathcal{L}^{\mathcal{M}}_{k\cdot t}$. Assume that $\mathcal{L}_t$ has an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP $(\mathcal{P}_t, \mathcal{V}_t)$ w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, where the parameters satisfy the requirements in the lemma statement.

As discussed in the overview, the high-level idea for designing the PCIP $(\mathcal{P}_{t\cdot k}, \mathcal{V}_{t\cdot k})$ is for $\mathcal{P}_{t\cdot k}$ to first specify k evenly-spaced intermediate configurations of the Turing machine. Given these intermediate configurations, $\mathcal{V}_{t\cdot k}$ wants to verify that k statement are in $\mathcal{L}_t$, where the j^{th} statement refers to a computation of length t between two Turing machine configurations. Since $\mathcal{L}_t$ has an unambiguous PCIP, we can use our batch verification lemma to obtain an *efficient* PCIP $(\mathcal{P}_{\mathsf{B}}, \mathcal{V}_{\mathsf{B}})$ for verifying all k statements (the B stands for *Batched*). The latter PCIP has relatively many queries, so rather than running it directly, we first apply the query reduction transformation on it to derive a query-efficient PCIP $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ (where BQ stands for *Batched* and *Query Reduced*) and then have the verifier and prover emulate $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$.

The unambiguous PCIP for $\mathcal{L}_{k\cdot t}$, denoted by $(\mathcal{P}_{k\cdot t}, \mathcal{V}_{k\cdot t})$, is presented in Fig. 7.

Prescribed Completeness. Let $x \in \{0,1\}^n$, $u, v \in \{0,1\}^{O(S)}$. First, note that since the *prescribed* prover sends $\mathsf{LDE}(\tilde{w}_t, \ldots, \tilde{w}_{(k-1)\cdot t})$ such that $\tilde{w}_{j\cdot t} = w_{j\cdot t}$ for every $j \in [k-1]$, where $w_{j\cdot t}$ denotes the configuration of the Turing machine $\mathcal{M}$ after $j\cdot t$ steps (on input x starting at configuration u). By construction, it holds that $\forall j \in [k-1]$, $(x, \tilde{w}_{(j-1)\cdot t}, \tilde{w}_{j\cdot t}) \in \mathcal{L}_t$. We denote $u = w_0 = \tilde{w}_0$ and $v = \tilde{w}_{k\cdot t}$.

Consider first the case that $(x, u, v) \in \mathcal{L}_{k\cdot t}$ (i.e., $v = \tilde{w}_{k\cdot t} = w_{k\cdot t}$). In this case, we have that $\Big(\big(x, \tilde{w}_{(j-1)\cdot t}, \tilde{w}_{j\cdot t}\big)_{j\in[k]}\Big) \in \mathcal{L}_t^{\otimes k}$. Thus, $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ are run on a k-tuple that belongs to $\mathcal{L}_t^{\otimes k}$ and so

Unambiguous PCIP $(\mathcal{P}_{k\cdot t}, \mathcal{V}_{k\cdot t})$ **for** $\mathcal{L}_{k\cdot t}$

Parameters: $\sigma \in (0,1)$ and $\lambda \geq 1$.
Prover's Input: $x \in \{0,1\}^n$ and configurations $u, v \in \{0,1\}^{O(S)}$.
Verifier Input: implicit access to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x,u,v)$.

1. The prover $\mathcal{P}_{k\cdot t}$ runs the Turing machine $\mathcal{M}$ starting at configuration u for $k \cdot t$ steps. Let w_j be the configuration of $\mathcal{M}$ after j steps, for every $j \in \{0, \ldots, k \cdot t\}$.

   $\mathcal{P}_{k\cdot t}$ sends $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_t, w_{2t}, \ldots, w_{(k-1)\cdot t})$ to $\mathcal{V}_{k\cdot t}$ (where the message is padded with zeros to the overall maximum message length in the protocol, see Remark 4.13).

2. The verifier $\mathcal{V}_{k\cdot t}$ receives[a] $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(\tilde{w}_t, \ldots, \tilde{w}_{(k-1)\cdot t})$ (which is allegedly equal to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_t, w_{2t}, \ldots, w_{(k-1)\cdot t})$) and checks the padding using the procedure in Remark 4.13.

   Define $\tilde{w}_0 \stackrel{\text{def}}{=} u$ and $\tilde{w}_{k\cdot T} \stackrel{\text{def}}{=} v$ and observe that, using the procedure in Proposition 3.8, the verifier $\mathcal{V}_{k\cdot t}$ has implicit access to $\mathsf{LDE}(\tilde{w}_0)$ and $\mathsf{LDE}(\tilde{w}_{k\cdot T})$.

3. Let $(\mathcal{P}_{\mathsf{B}}, \mathcal{V}_{\mathsf{B}})$ be the batched PCIP for the language $\mathcal{L}_t^{\otimes k}$, obtained by applying Lemma 8.1 to the language $\mathcal{L}_T$ (which has the PCIP $(\mathcal{P}_t, \mathcal{V}_t)$), w.r.t. parameter σ and security parameter λ.

   Let $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ be the query reduced PCIP (also for the language $\mathcal{L}_t^{\otimes k}$), obtained by applying Lemma 8.2 to the PCIP $(\mathcal{P}_{\mathsf{B}}, \mathcal{V}_{\mathsf{B}})$, w.r.t. parameter $\sigma' = \sigma/c$, where $c \geq 1$ is some sufficiently large constant.

   The prover $\mathcal{P}_{k\cdot t}$ and verifier $\mathcal{V}_{k\cdot t}$ run $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ on (implicit) input the k-tuple $\left(\mathsf{LDE}_{\mathbb{F},\mathbb{H}}\left(x, \tilde{w}_{(j-1)\cdot t}, \tilde{w}_{j\cdot t}\right)\right)_{j\in[k]}$, where $\mathcal{V}_{k\cdot t}$ uses the procedure in Proposition 3.8 to answer $\mathcal{V}_{\mathsf{BQ}}$'s input queries. If $\mathcal{V}_{\mathsf{BQ}}$ accepts then $\mathcal{V}_{k\cdot t}$ accepts and otherwise it rejects.

[a] Recall that we restrict our attention to encoded provers and so may assume that the received message *is* a low degree extension encoding of some (possibly incorrect) configurations.

Figure 7: Unambiguous PCIP $(\mathcal{P}_{k\cdot t}, \mathcal{V}_{k\cdot t})$ for $\mathcal{L}_{k\cdot t}$

by the prescribed completeness of $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ (which follows from Lemmas 8.1 and 8.2) the verifier $\mathcal{V}_{\mathsf{BQ}}$ accepts and so $\mathcal{V}_{k\cdot t}$ also accepts.

In the other case (i.e., $(x,u,v) \notin \mathcal{L}_{k\cdot t}$), it must be that $(x, \tilde{w}_{(k-1)\cdot t}, v) = (x, \tilde{w}_{(k-1)\cdot T}, \tilde{w}_{k\cdot t}) \notin \mathcal{L}_t$, and so $\left(\left(x, \tilde{w}_{(j-1)\cdot t}, \tilde{w}_{j\cdot t}\right)_{j\in[k]} \right) \notin \mathcal{L}^{\otimes k}$. By the prescribed completeness of $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ (which follows from Lemmas 8.1 and 8.2) the verifier $\mathcal{V}_{\mathsf{BQ}}$ rejects and so $\mathcal{V}_{k\cdot t}$ also rejects.

As for the queries that $\mathcal{V}_{k\cdot t}$ makes, in Step 2 it merely checks the padding, and so its queries are input-oblivious. In Step 3, the verifier runs $\mathcal{V}_{\mathsf{BQ}}$. By Lemmas 8.1 and 8.2, $\mathcal{V}_{\mathsf{BQ}}$ only makes input-oblivious queries.

Unambiguity. Let $\widetilde{\mathcal{P}}_{k\cdot t}$ be a $(g_{\mathsf{aug}}, \mathbb{H}, \mathbb{F})$-encoded cheating prover, where the parameter g_{aug} is specified below (and in the lemma's statement).

If $\widetilde{\mathcal{P}}_{k\cdot t}$ deviates in the first round (i.e., in Step 1), then either it deviates on the padding (in which case $\mathcal{V}_{k\cdot t}$ rejects with probability $1 - 2^{-\lambda}$, see Remark 4.13) or it sends a message[22] $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(\tilde{w}_t, \ldots, \tilde{w}_{(k-1)\cdot t})$ such that there exists $j^* \in [k-1]$ for which $\tilde{w}_{j^*\cdot t} \neq w_{j^*\cdot t}$. That is, $\tilde{w}_{j^*\cdot t}$ is not the correct configuration of the (deterministic) Turing machine $\mathcal{M}$ after $j^* \cdot t$ steps, and so $(x, \tilde{w}_{(j^*-1)\cdot t}, \tilde{w}_{j^*\cdot t}) \notin \mathcal{L}_t$. Therefore, $\left(\mathsf{LDE}\left(x, \tilde{w}_{(j-1)\cdot t}, \tilde{w}_{j\cdot t}\right)\right)_{j\in[k]} \notin \mathcal{L}_t^{\otimes k}$. In Step 3, if $\widetilde{\mathcal{P}}_{k\cdot t}$ follows the prescribed protocol $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$, then, since it is run on a k-tuple that is not in $\mathcal{L}_t^{\otimes k}$, by the prescribed completeness of $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$, the verifier $\mathcal{V}_{amoQR}$ rejects and so also $\mathcal{V}_{k\cdot t}$ rejects.

Hence, we may assume that $\widetilde{\mathcal{P}}_{k\cdot t}$ deviates in some round $i^* \in \{2, \ldots, \ell_{\mathsf{aug}}\}$ (in addition to possibly deviating in the first round). That is, it is deviating within the protocol $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ which is an $\varepsilon_{\mathsf{BQ}}$-unambiguous PCIP (for a value $\varepsilon_{\mathsf{BQ}}$ that will be specified below). Therefore, $\mathcal{V}_{\mathsf{BQ}}$ (and hence also $\mathcal{V}_{k\cdot t}$) rejects with probability $\varepsilon_{\mathsf{BQ}}$.

Complexity Measures. Recall that we assumed that:

1. $\log(\ell^{1/\sigma} \cdot a) \leq \min\left(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\right)$.
2. $a \geq \max\left(\mathsf{poly}(k, S), \mathsf{poly}(k, \mathcal{V}\mathsf{time}, g) \cdot (\mathsf{poly}(\lambda, |\mathbb{H}|, \ell))^{1/\sigma}\right)$.
3. $\mathcal{P}\mathsf{time} \geq \ell \cdot a \cdot \mathsf{polylog}(|\mathbb{F}|)$.
4. $\mathcal{V}\mathsf{time} \geq \max(q, b \cdot \ell)$.

and so the PCIP $(\mathcal{P}_t, \mathcal{V}_t)$ satisfies the requirements of Lemma 8.1 and we obtain that the protocol $(\mathcal{P}_{\mathsf{B}}, \mathcal{V}_{\mathsf{B}})$ is an ε_{B}-unambiguous $(q_{\mathsf{B}}, \ell_{\mathsf{B}}, a_{\mathsf{B}}, b_{\mathsf{B}}, \mathcal{P}\mathsf{time}_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}})$-$\mathsf{PCIP}$ for the language $\mathcal{L}_T^{\otimes k}$ w.r.t. $(g_{\mathsf{B}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and the following parameters:

- $\varepsilon_{\mathsf{B}} = (\frac{1}{\sigma} + 1) \cdot \left(\varepsilon + 3 \cdot 2^{-\lambda}\right)$.
- $q_{\mathsf{B}} = k^2 \cdot q \cdot \mathsf{poly}(g) \cdot \left(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\right)^{1/\sigma}$.
- $\ell_{\mathsf{B}} = O(\ell/\sigma)$.
- $a_{\mathsf{B}} = a \cdot 2k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}$.

[22] Here we use the fact that $\widetilde{\mathcal{P}}_{k\cdot t}$ is a $(g_{\mathsf{aug}}, \mathbb{H}, \mathbb{F})$-encoded prover and so the message that it sends must be a low degree extension encoding of some (possibly incorrect) configurations.

- $b_{\mathsf{B}} = \max\big(b, k \cdot \mathsf{poly}(|\mathbb{H}|, \ell^{1/\sigma}, g, \lambda)\big)$.
- $\mathcal{P}\mathsf{time}_{\mathsf{B}} = 2k \cdot 1/\sigma \cdot \ell^{1/\sigma} \cdot \mathcal{P}\mathsf{time} + \mathcal{V}\mathsf{time} \cdot q \cdot k^2 \cdot \mathsf{poly}(g)\big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{1/\sigma}$.
- $\mathcal{V}\mathsf{time}_{\mathsf{B}} = \mathcal{V}\mathsf{time} \cdot q \cdot k^2 \cdot \mathsf{poly}(g) \cdot \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{1/\sigma}$.
- $g_{\mathsf{B}} = g \cdot 2k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}$.

By our assumptions that:

1. $|\mathbb{H}| \geq \log\Big(\mathsf{poly}\big(k, n, S, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time}\big) \cdot \big(\mathsf{poly}(\lambda, |\mathbb{H}|, \ell)\big)^{1/\sigma}\Big)$, and
2. $\mathcal{P}\mathsf{time} \geq \ell \cdot a \cdot \mathsf{polylog}(|\mathbb{F}|)$

it holds that

$$\log\big(\max(n_{\mathsf{B}}, \ell_{\mathsf{B}}, a_{\mathsf{B}}, b_{\mathsf{B}}, \mathcal{P}\mathsf{time}_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}})\big) \leq |\mathbb{H}| \tag{13}$$

where $n_{\mathsf{B}} = n + O(k \cdot S)$ is the relevant input length for the language $\mathcal{L}_t^{\otimes k}$. Also, $q_{\mathsf{B}} = k^2 \cdot q \cdot \mathsf{poly}(g) \cdot \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{1/\sigma} \geq (|\mathbb{H}|)^{\sigma'}$ (where $\sigma' = \sigma/c$ is the parameter used for the query reduction transformation in the protocol, see Step 3 in Fig. 7) and so $\min(q_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}})^{1/\sigma'} \geq q_{\mathsf{B}}^{1/\sigma'} \geq |\mathbb{H}|$. Hence, the PCIP $(\mathcal{P}_{\mathsf{B}}, \mathcal{V}_{\mathsf{B}})$ satisfies the requirements of Lemma 8.2. We conclude that $(\mathcal{P}_{\mathsf{BQ}}, \mathcal{V}_{\mathsf{BQ}})$ is an $\varepsilon_{\mathsf{BQ}}$-unambiguous $(q_{\mathsf{BQ}}, \ell_{\mathsf{BQ}}, a_{\mathsf{BQ}}, b_{\mathsf{BQ}}, \mathcal{P}\mathsf{time}_{\mathsf{BQ}}, \mathcal{V}\mathsf{time}_{\mathsf{BQ}})$-PCIP for the language $\mathcal{L}_t^{\otimes k}$ w.r.t. $(g_{\mathsf{BQ}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and the following parameters (recall that $\sigma' = \sigma/c$ for a sufficiently large constant c):

1.
$$\begin{aligned} q_{\mathsf{BQ}} &= \mathsf{poly}(\mathcal{V}\mathsf{time}_{\mathsf{B}})^{\sigma'} + O(\ell_{\mathsf{B}} \cdot g_{\mathsf{B}} \cdot \log|\mathbb{F}|) \\ &= (\mathcal{V}\mathsf{time} \cdot k \cdot g)^{\sigma} \cdot \mathsf{poly}(\ell, \lambda, |\mathbb{H}|) + O(k^{\sigma} \cdot \ell^{O(1/\sigma)} \cdot g \cdot \log(k) \cdot \lambda \cdot \log|\mathbb{F}| \cdot 1/\sigma) \end{aligned}$$

2.
$$\ell_{\mathsf{BQ}} = \ell_{\mathsf{B}} + O(1/\sigma') = O(\ell/\sigma)$$

3.
$$a_{\mathsf{BQ}} = \max\Big(a_{\mathsf{B}}, \mathsf{poly}(\ell_{\mathsf{B}}, b_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}}, |\mathbb{H}|)\Big) = \Big(2a \cdot k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}\Big)$$

4.
$$b_{\mathsf{BQ}} = \max\left(b_{\mathsf{B}}, O(|\mathbb{H}| \cdot \log|\mathbb{F}|)\right) = \max\Big(b, k \cdot \mathsf{poly}(|\mathbb{H}|, \ell^{1/\sigma}, g, \lambda)\Big)$$

5.
$$\begin{aligned} \mathcal{P}\mathsf{time}_{\mathsf{BQ}} &= \mathcal{P}\mathsf{time}_{\mathsf{B}} + \mathsf{poly}(q_{\mathsf{B}}, \ell_{\mathsf{B}}, b_{\mathsf{B}}, \mathcal{V}\mathsf{time}_{\mathsf{B}}) \\ &= 2k \cdot \mathcal{P}\mathsf{time} \cdot 1/\sigma \cdot \ell^{1/\sigma} + \mathsf{poly}(\mathcal{V}\mathsf{time}, k, g, \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{1/\sigma}) \end{aligned}$$

6.

$$\begin{aligned}\mathcal{V}\mathsf{time}_{\mathsf{BQ}} &= \mathsf{poly}(\mathcal{V}\mathsf{time}_{\mathsf{B}})^{\sigma'} + (\mathsf{poly}(b_{\mathsf{B}}, \ell_{\mathsf{B}}, g, |\mathbb{H}|)) \\ &= (\mathcal{V}\mathsf{time} \cdot k \cdot g)^{\sigma} \cdot \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big) + \mathsf{poly}(k, b, |\mathbb{H}|, \ell^{1/\sigma}, g, \lambda)\end{aligned}$$

7.

$$\varepsilon_{\mathsf{BQ}} = \varepsilon_{\mathsf{B}} + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|} = \left(\frac{1}{\sigma} + 1\right) \cdot \left(\varepsilon + 3 \cdot 2^{-\lambda}\right) + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}$$

8.

$$g_{\mathsf{BQ}} = g_{\mathsf{B}} = g \cdot 2k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}$$

Note that by Proposition 3.8, for every $j \in [k]$, every query to $\mathsf{LDE}\left(x, \tilde{w}_{(j-1)\cdot T}, \tilde{w}_{j\cdot T}\right)$ that the verifier $\mathcal{V}_{\mathsf{BQ}}$ makes can be emulated $\mathcal{V}_{k\cdot t}$ by making $O(1)$ queries to the implicit input $\mathsf{LDE}(x, u, v)$ and the message $\mathsf{LDE}(\tilde{w}_T, \ldots, \tilde{w}_{(k-1)\cdot T})$ sent by the prover in the first round. This also adds an overhead of $\mathsf{poly}(|\mathbb{H}|)$ in verifier computation per query.

Accounting also for the prover $\mathcal{P}_{k\cdot t}$'s and verifier $\mathcal{V}_{k\cdot t}$'s complexity in Step 1, overall we obtain that the protocol $(\mathcal{P}_{k\cdot t}, \mathcal{V}_{k\cdot t})$ is an $\varepsilon_{\mathsf{aug}}$-unambiguous $(q_{\mathsf{aug}}, \ell_{\mathsf{aug}}, a_{\mathsf{aug}}, b_{\mathsf{aug}}, \mathcal{P}\mathsf{time}_{\mathsf{aug}}, \mathcal{V}\mathsf{time}_{\mathsf{aug}})$-PCIP for the language $\mathcal{L}_T^{\otimes k}$ w.r.t. $(g_{\mathsf{aug}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and the following parameters:

1.

$$\begin{aligned}q_{\mathsf{aug}} &= O(q_{\mathsf{BQ}}) + \ell \cdot g_{\mathsf{BQ}} \cdot \mathsf{poly}(|\mathbb{H}|) \\ &= \Big(\big(\mathcal{V}\mathsf{time} \cdot k \cdot g\big)^{\sigma} \cdot \mathsf{poly}(\ell, \lambda, |\mathbb{H}|) + O(k^{\sigma} \cdot \ell^{O(1/\sigma)} \cdot g \cdot \log(k) \cdot \lambda \cdot \mathsf{poly}(|\mathbb{H}|))\Big)\end{aligned}$$

2.

$$\ell_{\mathsf{aug}} = 1 + \ell_{\mathsf{BQ}} = O(\ell/\sigma)$$

3.

$$a_{\mathsf{aug}} = \max\left(\mathsf{poly}(k, S), a_{\mathsf{BQ}}\right) = \left(2a \cdot k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}\right)$$

4.

$$b_{\mathsf{aug}} = b_{\mathsf{BQ}} = \max\left(b, k \cdot \mathsf{poly}(|\mathbb{H}|, \ell^{1/\sigma}, g, \lambda)\right)$$

5.

$$\begin{aligned}\mathcal{P}\mathsf{time}_{\mathsf{aug}} &= k \cdot t + \mathsf{poly}(S, k) + \mathcal{P}\mathsf{time}_{\mathsf{BQ}} \\ &= \Big(k \cdot t + \mathsf{poly}(k, S) + O\big(k \cdot \mathcal{P}\mathsf{time} \cdot 1/\sigma \cdot \ell^{1/\sigma}\big) + \mathsf{poly}\big(\mathcal{V}\mathsf{time}, k, g, \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big)^{1/\sigma}\big)\Big)\end{aligned}$$

6.

$$\begin{aligned}\mathcal{V}\mathsf{time}_{\mathsf{aug}} &= \mathcal{V}\mathsf{time}_{\mathsf{BQ}} + q_{\mathsf{BQ}} \cdot \mathsf{poly}(|\mathbb{H}|) \\ &= \Big((\mathcal{V}\mathsf{time} \cdot k \cdot g)^{\sigma} \cdot \big(\mathsf{poly}(\ell, \lambda, |\mathbb{H}|)\big) + \mathsf{poly}(k, b, |\mathbb{H}|, \ell^{1/\sigma}, g, \lambda)\Big)\end{aligned}$$

7.

$$\varepsilon_{\mathsf{aug}} = \max\left(\varepsilon_{\mathsf{BQ}}, 2^{-\lambda}\right) = \left(\left(\frac{1}{\sigma} + 1\right) \cdot \left(\varepsilon + 3 \cdot 2^{-\lambda}\right) + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}\right)$$

8.

$$g_{\mathsf{aug}} = g_{\mathsf{BQ}} = \left(g \cdot 2k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell^{1/\sigma}\right)$$

This completes the proof of Lemma 8.3. □

8.2 An Unambiguous PCIP for Bounded Space Computations

In this section we construct an efficient unambiguous PCIPs w.r.t. encoded provers, for bounded space computation (Theorem 10). Our main results, which are efficient interactive proofs for bounded space computations (see Section 5) follow from Theorem 10 by using the transformations between the different variants of interactive proofs that were shown in Section 4.

Theorem 10 (unambiguous PCIP for Bounded Space w.r.t. Encoded Provers)**.** *Let $T = T(n)$ and $S = S(n)$ such that $n \leq T \leq \mathsf{exp}(n)$ and $\log(T) \leq S \leq \mathsf{poly}(n)$. Let $\delta = \delta(n) \in (0, 1/2)$ be a parameter such that $\mathsf{poly}(1/\delta) \leq \log(T)$. Let $\mathbb{H} = (\mathbb{H}_n)_{n\in\mathbb{N}}$ and $\mathbb{F} = (\mathbb{F}_n)_{n\in\mathbb{N}}$ be constructible ensembles of finite fields such that $\mathbb{F}$ is an extension field of $\mathbb{H}$, where $|\mathbb{H}| = \log(T) \cdot (1/\delta)^{O(1/\delta)}$ and $|\mathbb{F}| = \mathsf{poly}(|\mathbb{H}|)$.*

Then, for every $\mathcal{L} \in \mathsf{DTISP}(T, S)$ there exists an ε-unambiguous $(q, \ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-PCIP w.r.t. $(g, \mathbb{H}, \mathbb{F})$-encoded provers (and with input-oblivious queries) with the following parameters:

- $\varepsilon = 1/\mathsf{polylog}(T)$.
- $q = T^{O(\delta^2)}$.
- $\ell = (1/\delta)^{O(1/\delta)}$.
- $a = T^{O(\delta)} \cdot \mathsf{poly}(S)$.
- $b = T^{O(\delta)}$.
- $\mathcal{P}\mathsf{time} = T^{1+O(\delta)} \cdot \mathsf{poly}(S)$.
- $\mathcal{V}\mathsf{time} = T^{O(\delta)}$.
- $g = T^{O(\delta^2)}$.

Proof. Fix T and S as above and let $\mathcal{M}$ be a time T and space S Turing machine for $\mathcal{L}$. We assume that $\mathcal{M}$ is an *oblivious* Turing machine, while noting that any time-T and space-S Turing machine can be simulated by a time $T' = O(T \cdot S)$ and space $S' = O(S)$ *oblivious*[23] Turing machine (and we shall account for this additional overhead at the end of the proof). Recall that a configuration $w \in \{0,1\}^{O(S)}$ of a Turing machine includes the contents of all work tapes, the current time step, the positions of the work and input heads and the current internal state. Let $w_{\mathsf{start}} \in \{0,1\}^{O(S)}$ be the machine $\mathcal{M}$'s initial configuration. We assume without loss of generality that $\mathcal{M}$ has a *unique* accepting configuration w_{end}. Furthermore, we assume without loss of generality that the configurations w_{start} and w_{end} are (fixed) strings such that individual points in their respective low degree extensions $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_{\mathsf{start}})$ and $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_{\mathsf{end}})$ can be evaluated in $\mathsf{poly}(\mathbb{H})$ time.[24]

Fix $k \stackrel{\text{def}}{=} T^{\delta}$. Recall that for every $t \leq T$, the language $\mathcal{L}_t$, defined in Section 8.1, consists of triplets $(x,u,v) \in \{0,1\}^n \times \{0,1\}^{O(S)} \times \{0,1\}^{O(S)}$ such that on input x, the Turing machine $\mathcal{M}$ moves from configuration u to configuration v in exactly t steps. We will show how to directly construct an unambiguous PCIP for $\mathcal{L}_1$ (i.e., $t = 1$). Then, using iterative applications of Lemma 8.3 we will obtain PCIPs for $\mathcal{L}_{(k^i)}$ for larger and larger values of $i \leq \log_k(T)$, until ultimately, when $i = \log_k(T)$, we obtain a PCIP for the language $\mathcal{L}_T$. An unambiguous PCIP for the language $\mathcal{L}$, as in the theorem's statement, follows by running the PCIP for $\mathcal{L}_T$ on implicit input $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x, w_{\mathsf{start}}, w_{\mathsf{end}})$, where input queries are emulated using queries to $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x)$, $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_{\mathsf{start}})$ and $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(w_{\mathsf{end}})$, as shown in Proposition 3.8.

We first construct a base PCIP for the language $\mathcal{L}_1$. This is essentially a PCIP for verifying single steps of the Turing machine. We remark that no interaction with a prover is necessary for this PCIP (i.e., the verifier can check membership in $\mathcal{L}_1$ by itself), but for technical convenience (specifically, to facilitate the application of Lemma 8.3) we will introduce some "dummy" interaction.

Proposition 8.4 (PCIP for $\mathcal{L}_1$)**.** *The language $\mathcal{L}_1$ has an ε_0-unambiguous $(q_0, \ell_0, a_0, b_0, \mathcal{P}\mathsf{time}_0, \mathcal{V}\mathsf{time}_0)$-$\mathsf{PCIP}$ $(\mathcal{P},\mathcal{V})$ w.r.t. $(g_0, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and parameters:*

- $q_0 = \mathsf{poly}(|\mathbb{H}|)$.
- $\ell_0 = 1$.
- $a_0 = \mathsf{poly}(k, S)$.
- $b_0 = 0$.
- $\mathcal{P}\mathsf{time}_0 = \mathsf{poly}(k, S)$.
- $\mathcal{V}\mathsf{time}_0 = \mathsf{poly}(|\mathbb{H}|)$.
- $\varepsilon_0 = \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}$.

[23] An $\mathsf{oblivious}$ Turing machine is a Turing machine whose input and work tape head *positions* are a fixed function of the current time step (and in particular do not depend on the input). Furthermore, given a time step $t \in [T]$, the head positions in time t can be computed in $\mathsf{polylog}(T)$ time. We remark that more efficient simulation is known [PF79].

[24] We can assume without loss of generality that both the initial configuration w_{start} and (unique) accepting configuration w_{end} are zero on all but $O(\log(T))$ bits (at fixed locations) and the values of the non-zero bits can be computed by a Turing machine in $O(\log(T))$ time. Each point in their respective low degree extension is therefore a linear combination (over the field $\mathbb{F}$) of these $O(\log(T))$ bits. The coefficients of this linear combination (and therefore also its sum) can be computed in time $\mathsf{poly}(|\mathbb{H}|, \log_{|\mathbb{H}|}(S)) = \mathsf{poly}(\mathbb{H})$, see Section 3.2.

- $g_0 = 1$.

Proof. The verifier first reads the time step t_u that appears as part of the configuration u. Since $\mathcal{M}$ is *oblivious*, given t_u we can determine in time $\mathsf{polylog}(T)$ a set $\Omega \subseteq [O(S)]$, of size $|\Omega| = O(\log(T))$, of coordinates on which u and v may differ (i.e., the time step, the position of the heads, the current state and the $O(1)$ locations in the work tapes on which the heads are currently positioned). The verifier queries the coordinates in Ω and checks that they were updated correctly (i.e., in accordance with the Turing machine's specification). For all other coordinates we need to check that they are equal in u and v. Since the low degree extension is a linear code, $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(u) - \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(v) = \mathsf{LDE}_{\mathbb{F},\mathbb{H}}(u-v)$ and so we only need to check that $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(u-v)$ is zero on all points outside Ω. The latter can be done by running the procedure in Proposition 3.10, in $\mathsf{poly}(|\mathbb{H}|)$ time, with an error probability of at most $\frac{O(|\mathbb{H}| \cdot \log_{|\mathbb{H}|}(S))}{|\mathbb{F}|} \leq \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}$.

For technical convenience, since we want a PCIP that satisfies the requirements of Lemma 8.3, we also have the prover send a single "dummy" message of length $\mathsf{poly}(k, S)$ (encoded using the low degree extension encoding). To maintain unambiguity, the verifier just checks that message sent from the prover is a low degree extension of the all-zeros string. Since we restrict our attention to encoded provers, it suffices to do so by checking that a random point in the low degree extension is zero, see Lemma 3.1.

□

We proceed to prove to the inductive step. Fix a security parameter $\lambda = O(\log(|\mathbb{F}|) + \log(1/\delta)) = O(\log(T))$.

Proposition 8.5. *Let $c \geq 1$ be a sufficiently large fixed constant. Then, for every $k = k(n) \geq 1$ and $i \in \{0, \ldots, \log_k(T)\}$, the language $\mathcal{L}_{(k^i)}$ has an ε_i-unambiguous $(q_i, \ell_i, a_i, b_i, \mathcal{P}\mathsf{time}_i, \mathcal{V}\mathsf{time}_i)$-PCIP $(\mathcal{P}, \mathcal{V})$ w.r.t. $(g_i, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and parameters:*

1. $q_i = \left((\mathcal{V}\mathsf{time}_0)^{\delta^{2i}} \cdot k^{O(\delta)} \cdot 2^{\mathsf{poly}(1/\delta)} \cdot \mathsf{poly}(\lambda^{1/\delta}, |\mathbb{H}|) \right)$.
2. $\ell_i = \left(c/\delta^2\right)^i$.
3. $a_i = \left(a_0 \cdot \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3} \right)^i \right)$.
4. $b_i = \left(k \cdot \mathsf{poly}\left(|\mathbb{H}|, \left(k^{\delta^2} \cdot \lambda \cdot (1/\delta)^{1/\delta^3} \right)^i \right) \right)$.
5. $\mathcal{P}\mathsf{time}_i = \left((i+1) \cdot k^i \cdot (1/\delta)^{O(i/\delta^3)} \cdot \mathsf{poly}\left(\mathcal{P}\mathsf{time}_0, k, S, (\mathcal{V}\mathsf{time}_0)^{\delta^{2/\delta}} \right) \cdot (\lambda \cdot |\mathbb{H}|)^{\mathsf{poly}(1/\delta)} \right)$.
6. $\mathcal{V}\mathsf{time}_i = \left((\mathcal{V}\mathsf{time}_0)^{\delta^{2i}} \cdot \left(\mathsf{poly}\left(k, |\mathbb{H}|, \lambda^{1/\delta}, (1/\delta^2)^{1/\delta^4} \right) \right)^{\sum_{j=0}^{i} \delta^{2j}} \right)$.
7. $\varepsilon_i = \left(\left(\frac{1}{\delta^2} + 1 \right)^i \cdot \left(\varepsilon_0 + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|} \right) \right)$.
8. $g_i = \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3} \right)^i$.

Proof. The proof is by induction. The base case $i = 0$ follows directly from Proposition 8.4.

Let $i \in \{1, \ldots, \log_k(T)\}$ and note that $i \leq \log_k(T) = 1/\delta$. Assume inductively that $\mathcal{L}_{(k^{i-1})}$ has an $\varepsilon_{(i-1)}$-unambiguous $(q_{(i-1)}, \ell_{(i-1)}, a_{(i-1)}, b_{(i-1)}, \mathcal{P}\mathsf{time}_{(i-1)}, \mathcal{V}\mathsf{time}_{(i-1)})$-PCIP $(\mathcal{P}_{(i-1)}, \mathcal{V}_{(i-1)})$ w.r.t. $(g_{(i-1)}, \mathbb{H}, \mathbb{F})$ and with input-oblivious queries, where the parameters are as in the proposition's statement (w.r.t. a sufficiently large fixed constant c).

We next show that the parameters of this PCIP satisfy the requirements of Lemma 8.3 with parameters k, $\sigma \stackrel{\text{def}}{=} \delta^2$ and security parameter λ (recall that we set $\lambda = O(\log(|\mathbb{F}|) + \log(1/\delta)) = O(\log(T))$ above). Indeed it holds that:
(note that here and below we extensively use our assumption that $\mathsf{poly}(1/\delta) \leq \log(T)$)

$$\begin{aligned}
\log\left(\ell_{(i-1)}^{1/\sigma} \cdot a_{(i-1)}\right) &= \log\left(\left(c/\delta^2\right)^{i/\sigma} \cdot \left(a_0 \cdot \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right)^i\right)\right) \\
&= \mathsf{poly}(1/\delta) \cdot \log(T) \\
&\leq \min\left(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\right)
\end{aligned}$$

$$\begin{aligned}
a_{(i-1)} &\geq a_0 \\
&= \mathsf{poly}\left(k, S\right) \\
&\geq \max\left(\mathsf{poly}(k, S), \mathsf{poly}(k, \mathcal{V}\mathsf{time}_{(i-1)}, g_{(i-1)}) \cdot \left(\mathsf{poly}(\lambda, |\mathbb{H}|, \ell_{(i-1)})\right)^{1/\sigma}\right)
\end{aligned}$$

$\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{(\text{follows by the fact that } k = T^{\delta})}$

$$\begin{aligned}
|\mathbb{H}| &= \log(T) \cdot (1/\delta)^{O(1/\delta)} \\
&\geq \log\left(\mathsf{poly}\big(k, n, S, \mathcal{P}\mathsf{time}_{(i-1)}, \mathcal{V}\mathsf{time}_{(i-1)}\big) \cdot \left(\mathsf{poly}(\lambda, |\mathbb{H}|, \ell_{(i-1)})\right)^{1/\sigma}\right)
\end{aligned}$$

$$\mathcal{P}\mathsf{time}_{(i-1)} \geq \ell_{(i-1)} \cdot a_{(i-1)} \cdot \mathsf{polylog}(|\mathbb{F}|)$$

$$\mathcal{V}\mathsf{time}_{(i-1)} \geq \max\left(q_{(i-1)}, b_{(i-1)} \cdot \ell_{(i-1)}\right)$$

By applying Lemma 8.3 with parameters as above, we conclude that the language $\mathcal{L}_{(k^i)}$ has an ε_i-unambiguous $(q_i, \ell_i, a_i, b_i, \mathcal{P}\mathsf{time}_i, \mathcal{V}\mathsf{time}_i)$-PCIP w.r.t. $(g_i, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and the following parameters:

1.
$$\begin{aligned}
q_i &= \left(\left(\mathcal{V}\mathsf{time}_{i-1} \cdot k \cdot g_{i-1}\right)^{\sigma} \cdot \mathsf{poly}(\ell_{i-1}, \lambda, |\mathbb{H}|) + O(k^{\sigma} \cdot \ell_{i-1}^{O(1/\sigma)} \cdot g_{i-1} \cdot \log(k) \cdot \lambda \cdot \mathsf{poly}(|\mathbb{H}|))\right) \\
&\leq \left(\mathcal{V}\mathsf{time}_{i-1}\right)^{\delta^2} \cdot k^{2\delta + 2\delta^2} \cdot (c/\delta^2)^{2/\delta^4} \cdot \mathsf{poly}(\lambda^{1/\delta}, |\mathbb{H}|) \\
&\leq \left(\mathcal{V}\mathsf{time}_0\right)^{\delta^{2i}} \cdot k^{O(\delta)} \cdot 2^{\mathsf{poly}(1/\delta)} \cdot \mathsf{poly}(\lambda^{1/\delta}, |\mathbb{H}|)
\end{aligned}$$

2.
$$\ell_i \leq c \cdot \ell_{i-1}/\sigma = c \cdot \left(c/\delta^2\right)^{i-1}/\delta^2 = \left(c/\delta^2\right)^i$$

3.

$$\begin{aligned}
a_i &= \left(2a_{i-1} \cdot k^{\sigma} \cdot \log(k) \cdot \lambda \cdot \ell_{i-1}^{1/\sigma}\right) \\
&\leq a_0 \cdot \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right)^{i-1} \cdot \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right) \\
&= a_0 \cdot \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right)^{i}
\end{aligned}$$

4.

$$b_i = \max\left(b_{i-1}, k \cdot \mathsf{poly}(|\mathbb{H}|, \ell_{i-1}^{1/\sigma}, g_{i-1}, \lambda)\right) = k \cdot \mathsf{poly}\left(|\mathbb{H}|, \left(k^{\delta^2} \cdot \lambda \cdot (1/\delta)^{1/\delta^3}\right)^{i}\right)$$

5.

$$\begin{aligned}
\mathcal{P}\mathsf{time}_i &= k^i + \mathsf{poly}(k, S) + O\big(k \cdot \mathcal{P}\mathsf{time}_{i-1} \cdot 1/\sigma \cdot \ell_{i-1}^{1/\sigma}\big) + \mathsf{poly}(\mathcal{V}\mathsf{time}_{i-1}, k, g_{i-1}, \big(\mathsf{poly}(\ell_{i-1}, \lambda, |\mathbb{H}|)\big)^{1/\sigma}) \\
&\leq \mathcal{P}\mathsf{time}_{i-1} \cdot k \cdot (1/\delta)^{O(1/\delta^3)} + k^i + \mathsf{poly}\left(k, S, (\mathcal{V}\mathsf{time}_0)^{\delta^{2/\delta}}\right) \cdot (\lambda \cdot |\mathbb{H}|)^{\mathsf{poly}(1/\delta)} \\
&\leq i \cdot k^i \cdot (1/\delta)^{O(i/\delta^3)} \cdot \mathsf{poly}\left(\mathcal{P}\mathsf{time}_0, k, S, (\mathcal{V}\mathsf{time}_0)^{\delta^{2/\delta}}\right) \cdot (\lambda \cdot |\mathbb{H}|)^{\mathsf{poly}(1/\delta)} \\
&\quad + k^i + \mathsf{poly}\left(k, S, (\mathcal{V}\mathsf{time}_0)^{\delta^{2/\delta}}\right) \cdot (\lambda \cdot |\mathbb{H}|)^{\mathsf{poly}(1/\delta)} \\
&\leq (i+1) \cdot k^i \cdot (1/\delta)^{O(i/\delta^3)} \cdot \mathsf{poly}\left(\mathcal{P}\mathsf{time}_0, k, S, (\mathcal{V}\mathsf{time}_0)^{\delta^{2/\delta}}\right) \cdot (\lambda \cdot |\mathbb{H}|)^{\mathsf{poly}(1/\delta)}
\end{aligned}$$

6.

$$\begin{aligned}
\mathcal{V}\mathsf{time}_i &= \Big((\mathcal{V}\mathsf{time}_{i-1} \cdot k \cdot g_{i-1})^{\sigma} \cdot \big(\mathsf{poly}(\ell_{i-1}, \lambda, |\mathbb{H}|)\big) + \mathsf{poly}(k, b_{i-1}, |\mathbb{H}|, \ell_{i-1}^{1/\sigma}, g_{i-1}, \lambda)\Big) \\
&\leq (\mathcal{V}\mathsf{time}_{i-1})^{\delta^2} \cdot \mathsf{poly}\left(k, |\mathbb{H}|, \lambda^{1/\delta}, (1/\delta^2)^{1/\delta^4}\right) \\
&\leq (\mathcal{V}\mathsf{time}_0)^{\delta^{2i}} \cdot \left(\mathsf{poly}\left(k, |\mathbb{H}|, \lambda^{1/\delta}, (1/\delta^2)^{1/\delta^4}\right)\right)^{\sum_{j=0}^{i} \delta^{2j}}
\end{aligned}$$

7.

$$\begin{aligned}
\varepsilon_i &= \left(\frac{1}{\sigma} + 1\right) \cdot \left(\varepsilon_{i-1} + 3 \cdot 2^{-\lambda}\right) + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|} \\
&= \left(\frac{1}{\delta^2} + 1\right) \cdot \varepsilon_{i-1} + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|} \\
&\leq \left(\frac{1}{\delta^2} + 1\right)^{i} \cdot \left(\varepsilon_0 + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}\right)
\end{aligned}$$

8.

$$\begin{aligned}
g_i &= \left(g_{i-1} \cdot 2k^{\sigma} \cdot \log(k) \cdot \lambda \cdot (\ell_{i-1})^{1/\sigma}\right) \\
&\leq \left(\left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right)^{i-1} \cdot 2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right) \\
&= \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right)^{i}
\end{aligned}$$

This completes the proof of Proposition 8.5. □

In particular, from Proposition 8.5 we obtain that the language $\mathcal{L}_{(k^{(1/\delta)})} = \mathcal{L}_T$ has an $\varepsilon_{(1/\delta)}$-unambiguous $(q_{(1/\delta)}, \ell_{(1/\delta)}, a_{(1/\delta)}, b_{(1/\delta)}, \mathcal{P}\mathsf{time}_{(1/\delta)}, \mathcal{V}\mathsf{time}_{(1/\delta)})$-PCIP w.r.t. $(g_{(1/\delta)}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, and the following parameters:

1. $q_{(1/\delta)} = (\mathcal{V}\mathsf{time}_0)^{\delta^{2/\delta}} \cdot k^{O(\delta)} \cdot 2^{\mathsf{poly}(1/\delta)} \cdot \mathsf{poly}(\lambda^{1/\delta}, |\mathbb{H}|) = T^{O(\delta^2)}$.
2. $\ell_{(1/\delta)} = \left(c/\delta^2\right)^{1/\delta} = (1/\delta)^{O(1/\delta)}$.
3. $a_{(1/\delta)} = a_0 \cdot \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right)^{(1/\delta)} = T^{O(\delta)} \cdot \mathsf{poly}(S)$.
4. $b_{(1/\delta)} = k \cdot \mathsf{poly}\left(|\mathbb{H}|, \left(k^{\delta^2} \cdot \lambda \cdot (1/\delta)^{1/\delta^3}\right)^{1/\delta}\right) = T^{O(\delta)}$.
5. $\mathcal{P}\mathsf{time}_{(1/\delta)} = (1/\delta) \cdot k^{1/\delta} \cdot (1/\delta)^{O(1/\delta^4)} \cdot \mathsf{poly}\left(\mathcal{P}\mathsf{time}_0, k, S, (\mathcal{V}\mathsf{time}_0)^{\delta^{2/\delta}}\right) \cdot (\lambda \cdot |\mathbb{H}|)^{\mathsf{poly}(1/\delta)} = T^{1+O(\delta)} \cdot \mathsf{poly}(S)$.
6. $\mathcal{V}\mathsf{time}_{(1/\delta)} = (\mathcal{V}\mathsf{time}_0)^{\delta^{2/\delta}} \cdot \left(\mathsf{poly}\left(k, |\mathbb{H}|, \lambda^{1/\delta}, (1/\delta^2)^{1/\delta^4}\right)\right)^{\sum_{j=0}^{(1/\delta)} \delta^{2j}} = T^{O(\delta)}$.
7. $\varepsilon_{(1/\delta)} = \left(\frac{1}{\delta^2} + 1\right)^{1/\delta} \cdot \left(\varepsilon_0 + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}\right) \leq \frac{1}{\mathsf{polylog}(T)}$.
8. $g_{(1/\delta)} = \left(2k^{\delta^2} \cdot \log(k) \cdot \lambda \cdot (c/\delta^2)^{1/\delta^3}\right)^{1/\delta} = T^{O(\delta^2)}$.

Theorem 10 follows by transforming the foregoing PCIP for $\mathcal{L}_T$ into a PCIP for $\mathcal{L}$ as explained above (i.e., by emulating queries to the input $\mathsf{LDE}_{\mathbb{F},\mathbb{H}}(x, w_{\mathsf{start}}, w_{\mathsf{end}}$ using Proposition 3.8).

To complete the proof, recall that we assumed that $\mathcal{M}$ is an *oblivious* Turing machine. In case it is not, we can trivially make it oblivious while increasing its time complexity to $T' = O(T \cdot S)$ and its space complexity to $S' = O(S)$ and the theorem follows. □

Applying Proposition 4.14 to the PCIP of Theorem 10 we obtain Theorem 7. Corollaries 8 and 9 follow from Theorem 7 as shown in Section 5.

9 Batch Verification of Unambiguous Interactive Proofs

In this section we show a batch verification lemma for *unambiguous interactive proofs*, based on the batch verification lemma for unambiguous PCIPs w.r.t. encoded provers (Lemma 8.1). We note that to prove our main results (i.e., interactive proofs for bounded space computations) we use Lemma 8.1 directly. We include the following batch verification lemma since we believe it may be of independent interest.

Theorem 11 (Batch Verification Theorem for Unambiguous IPs)**.** *Let $k = k(n) \geq 1$ and $\sigma = \sigma(n) > 0$. Suppose that the language $\mathcal{L}$ has an ε-unambiguous $(\ell, a, b, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})$-IP $(\mathcal{P}, \mathcal{V})$ over the alphabet $\{0,1\}$. We assume that:*

1. $\mathcal{P}\mathsf{time} \geq \mathcal{V}\mathsf{time} \geq \max\left(\ell \cdot (a+b), n\right)$.

2. $\sigma = \sigma(n) > \max\left(\frac{1}{\log(\mathcal{P}\mathsf{time})}, \varepsilon\right)$.

3. $a \geq \mathsf{poly}(k, \mathcal{V}\mathsf{time}^{\sigma}) \cdot \left(\mathsf{poly}(\log(\mathcal{P}\mathsf{time}), \ell)\right)^{1/\sigma}$.

Then, there exists an ε_{A}*-unambiguous* $(\ell_{\mathsf{A}}, a_{\mathsf{A}}, b_{\mathsf{A}}, \mathcal{P}\mathsf{time}_{\mathsf{A}}, \mathcal{V}\mathsf{time}_{\mathsf{A}})$-$\mathsf{IP}$ *for the language* $\mathcal{L}^{\otimes k}$*, with the following parameters:*

- $\varepsilon_{\mathsf{A}} = O(\varepsilon/\sigma)$.
- $\ell_{\mathsf{A}} = O(\ell/\sigma)$.
- $b_{\mathsf{A}} = \max\left(b, k \cdot \mathsf{poly}(\log(\mathcal{P}\mathsf{time}), (\ell/\sigma)^{(1/\sigma)})\right)$.
- $a_{\mathsf{A}} = k^{\sigma} \cdot \mathsf{poly}(a, \mathcal{V}\mathsf{time}) \cdot (\ell/\sigma)^{(1/\sigma)}$.
- $\mathcal{P}\mathsf{time}_{\mathsf{A}} = k \cdot \mathcal{P}\mathsf{time} \cdot (\ell/\sigma)^{1/\sigma} + \mathsf{poly}\left(k, \mathcal{V}\mathsf{time}, \ell^{1/\sigma}, (\log(\mathcal{P}\mathsf{time}))^{1/\sigma}\right)$.
- $\mathcal{V}\mathsf{time}_{\mathsf{A}} = \left(k \cdot n + \mathcal{V}\mathsf{time}^{\sigma} \cdot k^2 \cdot \mathsf{poly}(b) + k^{\sigma} \cdot \mathsf{poly}(\mathcal{V}\mathsf{time})\right) \cdot \left(\mathsf{poly}(\ell, \log(\mathcal{P}\mathsf{time}))\right)^{1/\sigma}$.

Remark 9.1 (Batch Verification for Different Languages)**.** *Theorem 11 holds even w.r.t.* k different *languages* $\mathcal{L}_1, \ldots, \mathcal{L}_k$*, so long as they all have* UIP*s with the same parameters. The batched* UIP *verifies that* $\forall j \in [k], x_j \in \mathcal{L}_j$.

Remark 9.2 (Batch Verification using $\mathsf{IP} = \mathsf{PSPACE}$)**.** *In retrospect, the fact that we can batch verify interactive proofs should not come as a surprise. An inefficient batch verification already follows from the* $\mathsf{IP} = \mathsf{PSPACE}$ *Theorem [LFKN92, Sha92], as described next. Suppose that we have* k *interactive proofs, each with communication* c*, that we want to batch verify. Each such protocol can be emulated by a Turing machine that runs in time* $2^{O(c)}$ *and space* $O(c)$*. Hence, all protocols together can be emulated by a* single *Turing machine that runs all of these computations in time roughly* $k \cdot 2^{O(c)}$ *and space* $O(c) + \log(k)$ *(see e.g. [GH98]). Applying the* $\mathsf{IP} = \mathsf{PSPACE}$ *Theorem to the latter Turing machine yields a batched interactive proof in which the communication and verifier running times are* $\mathsf{poly}(c, \log(k))$.

Note however that this batch verification is very inefficient in its round complexity and the honest prover's running time. More specifically, the number of rounds of interaction in this batch verification protocol is $\mathsf{poly}(c, \log(k))$*, regardless of the number of rounds in the base protocol, and the honest prover's running time is* $2^{\mathsf{poly}(c, \log k)}$*, also regardless of the prover's efficiency in the base protocol. In particular, this approach is not conducive towards our goals of obtaining a constant number of rounds and an efficient prover. The main advantage of Theorem 6 is in obtaining a batch verification that (roughly) preserves the round-complexity and the prover-efficiency of the base protocols.*

Proof of Theorem 11. Let $(\mathcal{P}, \mathcal{V})$ be an unambiguous IP for $\mathcal{L}$ as in the theorem's statement. Let $\mathbb{H}$ and $\mathbb{F}$ be constructible field ensembles such that $\mathbb{F}$ is an extension field of $\mathbb{H}$, where $|\mathbb{H}| = \Theta\left(\log(\mathcal{P}\mathsf{time})/\sigma\right) = \mathsf{poly}(\log(\mathcal{P}\mathsf{time}))$ and $|\mathbb{F}| = \mathsf{poly}(|\mathbb{H}|)$.

Observe that by our assumptions and our setting of $\mathbb{H}$, $\log(\max(n, \mathcal{P}\mathsf{time}, \mathcal{V}\mathsf{time})) \leq |\mathbb{H}| \leq \mathsf{poly}(n, \ell, a)^{\sigma}$. Hence, by Proposition 4.15, $\mathcal{L}$ has an ε'-unambiguous $(q', \ell', a', b', \mathcal{P}\mathsf{time}', \mathcal{V}\mathsf{time}')$-$\mathsf{PCIP}$ $(\mathcal{P}', \mathcal{V}')$ over the alphabet $\mathbb{F}$ w.r.t $(1, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, with the following parameters:

- $q' = \mathcal{V}\mathsf{time}^{\sigma} + O(\ell \cdot \log(|\mathbb{F}|))$.
- $\ell' = \ell + O(1/\sigma)$.
- $a' = \big(\mathsf{poly}(a, \ell, b, \mathcal{V}\mathsf{time}, |\mathbb{H}|)\big)$.
- $b' = \max\left(b, O(|\mathbb{H}| \cdot \log|\mathbb{F}|)\right)$.
- $\mathcal{P}\mathsf{time}' = \mathcal{P}\mathsf{time} + \mathsf{poly}(n, a, \ell, b, \mathcal{V}\mathsf{time}, |\mathbb{H}|)$.
- $\mathcal{V}\mathsf{time}' = \mathcal{V}\mathsf{time}^{\sigma} + (\mathsf{poly}(\ell, b, |\mathbb{H}|))$.
- $\epsilon' = \epsilon + \frac{\mathsf{poly}(|\mathbb{H}|)}{|\mathbb{F}|}$.

Fix $\lambda = \Theta(\log(|\mathbb{F}|))$. Observe that by our setting of parameters it holds that:

- $\log\left((\ell')^{1/\sigma} \cdot \mathsf{poly}(a', \ell', b', \mathcal{V}\mathsf{time}', |\mathbb{H}|)\right) \leq \min\left(|\mathbb{H}|, \frac{|\mathbb{F}|}{2|\mathbb{H}|}\right)$.
- $a' \geq \mathsf{poly}(k, q', g') \cdot (\mathsf{poly}(\lambda, |\mathbb{H}|, \ell'))^{1/\sigma}$.
- $\mathcal{P}\mathsf{time}' \geq \ell' \cdot a' \cdot \mathsf{polylog}(|\mathbb{F}|)$.

Thus, we can apply Lemma 8.1, with respect to parameters k and σ and security parameter λ to obtain that $\mathcal{L}^{\otimes k}$ has an ε_{A}-unambiguous $(q_{\mathsf{A}}, \ell_{\mathsf{A}}, a_{\mathsf{A}}, b_{\mathsf{A}}, \mathcal{P}\mathsf{time}_{\mathsf{A}}, \mathcal{V}\mathsf{time}_{\mathsf{A}})$-PCIP w.r.t. $(g_{\mathsf{A}}, \mathbb{H}, \mathbb{F})$-encoded provers and with input-oblivious queries, with the following parameters:

- $\varepsilon_{\mathsf{A}} = O(\varepsilon/\sigma)$.
- $q_{\mathsf{A}} = k^2 \cdot \mathcal{V}\mathsf{time}^{\sigma} \cdot \big(\mathsf{poly}(\ell, \log(\mathcal{P}\mathsf{time}))\big)^{1/\sigma}$.
- $\ell_{\mathsf{A}} = O(\ell/\sigma)$.
- $b_{\mathsf{A}} = \max\big(b, k \cdot \mathsf{poly}(\log(\mathcal{P}\mathsf{time}), (\ell/\sigma)^{(1/\sigma)})\big)$.
- $a_{\mathsf{A}} = k^{\sigma} \cdot \mathsf{poly}(a, \mathcal{V}\mathsf{time}) \cdot (\ell/\sigma)^{(1/\sigma)}$.
- $\mathcal{P}\mathsf{time}_{\mathsf{A}} = k \cdot \mathcal{P}\mathsf{time} \cdot (\ell/\sigma)^{1/\sigma} + \mathsf{poly}\left(k, \mathcal{V}\mathsf{time}, \ell^{1/\sigma}, (\log(\mathcal{P}\mathsf{time}))^{1/\sigma}\right)$.
- $\mathcal{V}\mathsf{time}_{\mathsf{A}} = \mathcal{V}\mathsf{time}^{2\sigma} \cdot k^2 \cdot \mathsf{poly}(b) \cdot \big(\mathsf{poly}(\ell, \log(\mathcal{P}\mathsf{time}))\big)^{1/\sigma}$.
- $g_{\mathsf{A}} = k^{\sigma} \cdot \log(k) \cdot (\ell/\sigma)^{O(1/\sigma)} \cdot \mathsf{polylog}(\log(\mathcal{P}\mathsf{time}))$.

Theorem 11 follows by applying Proposition 4.14. □

Acknowledgments

We thank Oded Goldreich, Shafi Goldwasser and Yael Kalai for invaluable and insightful conversations and comments. We also thank Avi Wigderson for pointing out the extension to delegating randomized computations (see Section 1.2).

Lastly, we thank the anonymous SICOMP reviewers for useful comments and suggestions.

References

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[AS03] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. *Combinatorica*, 23(3):365–426, 2003.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120, 2013.

[BCG+16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Short interactive oracle proofs with constant query complexity, via composition and sumcheck. Cryptology ePrint Archive, Report 2016/324, 2016. http://eprint.iacr.org/.

[BCGV16] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasi-linear size zero knowledge from linear-algebraic pcps. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, pages 33–64, 2016.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 31–60, 2016.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31, 1991.

[BGG+88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, pages 37–56, 1988.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 439–448, 2015.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 474–482, 2017.

[BM88] László Babai and Shlomo Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[CD97] Ronald Cramer and Ivan Damgård. Linear zero-knowledge - A note on efficient zero-knowledge proofs and arguments. In *STOC*, pages 436–445, 1997.

[CFLS95] Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter W. Shor. Probabilistically checkable debate systems and nonapproximability of PSPACE-hard functions. *Chicago J. Theor. Comput. Sci.*, 1995, 1995.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for RAM programs. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 429–437, 2015.

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.

[Din07] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

[DR06] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

[Dru11] Andrew Drucker. Efficient probabilistically checkable debates. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 519–529, 2011.

[Dru15] Andrew Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015.

[EKR04] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Inf. Comput.*, 189(2):135–159, 2004.

[FK97] Uriel Feige and Joe Kilian. Making games short (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 506–516, 1997.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 626–645, 2013.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

[GGR15] Oded Goldreich, Tom Gur, and Ron D. Rothblum. Proofs of proximity for context-free languages and read-once branching programs - (extended abstract). In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 666–677, 2015.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, 1998.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27, 2015.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

[Gol11] Oded Goldreich. Bravely, moderately: A common theme in four recent works. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 373–389. 2011.

[Gol16] Oded Goldreich. Lecture notes on low degree tests, 2016.

[GR15] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 133–142, 2015.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Inf. Process. Lett.*, 43(4):169–174, 1992.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and PCPs of almost-linear length. *J. ACM*, 53(4):558–655, 2006.

[GVW02] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1-2):1–53, 2002.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[HN10] Danny Harnik and Moni Naor. On the compressibility of *NP* instances and cryptographic applications. *SIAM J. Comput.*, 39(5):1667–1713, 2010.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 21–30, 2007.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 419–428, 2015.

[KR08] Yael Tauman Kalai and Ran Raz. Interactive PCP. In *ICALP*, pages 536–547, 2008.

[KR09] Yael Kalai and Guy N. Rothblum. Constant-round interactive proofs for NC^1. Unpublished observation, 2009.

[KR15] Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity - [extended abstract]. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 422–442, 2015.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *STOC*, pages 565–574, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 485–494, 2014.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, pages 169–189, 2012.

[Mei13] Or Meir. IP = PSPACE using error-correcting codes. *SIAM J. Comput.*, 42(1):380–403, 2013.

[Mei16] Or Meir. Combinatorial pcps with short proofs. *Comput. Complex.*, 25(1):1–102, 2016.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[Nis92] Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, pages 283–315, 2017.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 194–203, 1994.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

[Rot09] Guy N. Rothblum. *Delegating computation reliably: paradigms and constructions*. PhD thesis, Massachusetts Institute of Technology, 2009.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016.

[RRR18] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Efficient batch verification for UP. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 22:1–22:23, 2018.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[RVW00] Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 3–13, 2000.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *STOC*, pages 793–802, 2013.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer, 1995.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015.

[Wol65] Jack K. Wolf. On codes derivable from the tensor product of check matrices. *IEEE Trans. Information Theory*, 11(2):281–284, 1965.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il